**Вопросы к экзамену**

**по дисциплине**

**"Распределенные и облачные системы"**

1. Основные характеристики распределенных ИС: коллекция автономных вычислительных элементов, оверлейные сети; единая согласованная система.

Характеристика 1: Коллекция автономных вычислительных элементов (1)

* Современные РС содержат различные виды вычислительных узлов, в качестве своих компонентов. При этом действует фундаментальный принцип:

«Все узла могут действовать независимо друг от друга»

Характеристика 1: Коллекция автономных вычислительных элементов (2)

* Результатом независимости узлов являются:
  + Независимость времени – у каждого узла свое представление о текущем времени. Возникает проблема синхронизации и координации работы узлов;
  + Возможность организации группового управления коллекцией узлов, при этом могут образовываться открытые и закрытые группы. Групповое управление обеспечивает эффективность управления информационной безопасностью;

Характеристика 1: Коллекция автономных вычислительных элементов (3)

* + В результате организации коллекций на практике часто возникают т.н. оверлейные сети (покрывающие сети). При этом могут встречаться 2 вида перекрытия:
  + Структурное – в этом случае узлы хорошо осведомлены о своих соседях с которыми они могут поддерживать коммуникации.
  + Не структурное (без структурное) – в этом случае каждый узел имеет определенное число ссылок на соседей, но выбирает их случайным образом.
  + Оверлейные сети всегда связные, т.е. между двумя любыми узлами всегда имеется путь.
  + Примером оверлейных сетей являются сети Р2Р (peer-to-peer)

Характеристика 2: Единая согласованная система (1)

* Представление РС как единой системы порождает у пользователя ощущение «прозрачности» системы.
* С другой стороны отказ одного из узлов РС может привести к отказу всей системы в целом, что порождает проблему надежности и восстановления после отказов.

1. История возникновения и развития распределенных ИС. Эволюция РИС.

1960-е – Появление ОС, реализующих межпроцессное взаимодействие

1960-е – ARPANET и далее Usenet, FidoNet, Internet…

1960-1970-е – появление многопроцессорных систем

1970-е – Широкое распространение ЛВС на базе Ethernet

1980-е – Распределенные вычисления становятся активно изучаемой и обсуждаемой темой

2000-е – Корпоративные ИС повсеместно становятся распределенными.

Эволюция распределенных систем

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Эволюция систем HPC и HTC.

HPC

* В сегменте HPC суперкомпьютеры (массово-параллельные процессоры или MPP) постепенно заменяются кластерами совместно работающих компьютеров формирующих вычислительные ресурсы.
* Кластер часто представляет собой набор однородных вычислительных узлов, которые физически соединены сетью и располагаются на близком расстоянии друг от друга.
* Кластер в отличии от Р2Р систем всегда имеет управляющий узел.

HТC

* В сегменте HTC формируются одноранговые (P2P) сети для распределенного обмена файлами и приложений доставки контента. P2P-система строится на множестве клиентских машин. Одноранговые машины по своей природе распределены глобально.
* P2P, облачные вычисления и платформы веб-сервисов больше ориентированы на приложения HTC, чем на приложения HPC.
* Кластеризация и технологии P2P явились основой для создания вычислительных грид и/или грид данных.

Эволюция распределенных систем

* Типы современных РС сформировались в процессе эволюции двух классов систем:
* Систем высопроизводительных вычислений (HPC – High Performance Computing) области применения – наука и инженерия;
* Систем высокой пропускной способности (HTC – High Throughput Computing) – бизнес и управление;

1. Классификация распределенных ИС. Преимущества и недостатки РИС. Особенности современных РИС.

* **По архитектуре** 
  + –Клиент-серверная (client-server)
  + –3-tier / N-tier
  + –Одноранговая (peer-to-peer)
  + –Кластерная (сильно связанная)
  + –С общим адресным пространством
* **По территории** 
  + –Локальная
  + –Городская
  + –Глобальная
* **По степени автоматизации** 
  + –Автоматизированные
  + –Автоматические
* **По способу обработки данных** 
  + –Информационно-справочные
  + –Вычислительные (обработки данных)
* **По наличию центрального компонента** 
  + –С центральным компонентом
  + –Без центрального компонента
* **По реактивности** 
  + –Реального времени
  + –Без жестких требований ко времени реакции

Преимущества РИС

* Получения доступа к физически недоступным ресурсам
* Увеличение производительности системы по сравнению с единым локальным приложением
* Синергетические выгоды – самоорганизация в РС
* Высокая надежность, за сет дублирования всех важнейших компонентов.

Недостатки РИС

* Увеличенное время реакции системы
* Сложность контроля удаленных элементов
* Сложность разработки, отладки и использования
* Дополнительные усилия по обеспечению информационной безопасности
* Низкая надежность

**Особенности современных РИС**

* Сложность
  + –Описания
  + –Структуры
* Наличие подсистем
* Уникальные ограничения – невозможно использования без изменений типовые проектные решения
* Доработка существующих РИС – необходимость интеграции существующих приложений

Особенности современных РИС

* Многоплатформенность
* Совместная работа нескольких коллективов
* Длительность
  + –Разработки
  + –Существования РИС

1. Состав РИС. Виды обеспечения РИС. Корпоративные РИС.

Состав РИС

* Хосты / компьютеры
* Сеть / средства коммуникации
* Приложения
* Данные
* Пользователи

Виды обеспечения

* Информационное
* Техническое
* Математическое и программное
  + –ОПО
  + –СПО
* Организационное
* Правовое

Корпоративная ИС = РИС



1. Основные задачи создания и свойства РИС: доступность; прозрачность; открытость; гибкость; масштабируемость РИС.

Цели создания РИС

* Поддержка распределенного доступа к ресурсам
* Обеспечение прозрачности
* Обеспечение открытости
* Обеспечение масштабируемости
* Обеспечение безотказности работы

Основные задачи РИС

* Обеспечить связь пользователей с ресурсами РИС
  + **Ресурсы**  **распределенных информационных систем**: вычислительные ресурсы (процессорное время), информационные ресурсы (файлы, базы данных), принтеры и пр.
* Обеспечить прозрачность РИС
  + Системы которые представляются пользователям и приложениям в виде единой системы называются прозрачными.
* Обеспечить открытость РИС
  + Системы построенные на открытых стандатрах называются открытыми
* Обеспечить масштабируемость РИС
  + Системы допускающие рост ее размеров без потери свой производительности называется масштабируемой.

**Задача** 1. Доступ пользователей к ресурсам РИС

* Это основная задача РИС. Она решается средствами сетевых технологий.
* Решение ее позволяет организовать совместную работу пользователей с ресурсами и друг с другом в рамках решения общих задач.
* Возникли системы совместной (групповой работы), системы электронной коммерции, электронного образования (дистанционного).
* Совместный доступ к ресурсам усложнил решение задач обеспечения информационной безопасности.

**Задача** 2. Прозрачность РИС

* Прозрачность – это свойство РИС когда она представляется пользователю, как нечто единое целое.
* Концепция прозрачности применима к следующим аспектам РИС:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Степень прозрачности

* Существуют объективные ограничения на обеспечение полной прозрачности.
* Например:
  + Ограничения на скорость передачи по сети;
  + Ограничения на производительность системы;
  + Ограничения на скрытие временной рассинхронизации из-за конечного времени выполнения репликации.

Прозрачность: за и против

* Достоинства:
  + –Очевидным образом дополняет масштабируемость РИС
  + –Унификация пользовательских интерфейсов облегчает обучение пользователей и дальнейшее использование
  + –Унификация интерфейсов программ облегчает разработку (см. открытость далее)
* Недостатки
  + –Дополнительные затраты на реализацию
  + –Возможная неэффективность решений в каждом конкретном случае
  + –Наследование «тяжелого» интерфейса от уже исчезнувших ОС и техники
  + –Навязывание старых принципов организации работы и бизнес-процессов
  + –Усложнение обеспечения безопасности

**Задача** 3. Открытость

* Обеспечивает:
  + интероперабельность - способность к взаимодействию
  + переносимость – способность к переносу приложений между разными РИС.
* Основывается на использовании при проектировании ИС открытых стандартов, спецификаций и протоколов, определяющих порядок взаимодействия между всех компонентов распределенной системы между собой, а также пользователей с распределенной системой.
* Реализует разделение правил выполнения взаимодействий от механизмов их реализующих.

Уровни достижения открытости

* Унификация может достигаться на разных уровнях:
  + Синтаксическом
    - Одинаковые форматы данных
  + Семантическом
    - Одинаковые единицы измерения
    - Общая модель данных
    - Одинаковый API
  + Прагматическом
    - Общее понимание логики действий компонентов и РИС в целом
    - Одинаковый подход к интерпретации данных
  + Структурном
    - Использование однотипных аппаратных средств (архитектуры процессоров, сетевое оборудование и т.п.)
    - Использование однотипных программных средств (виды ОС, СУБД, ПО Web серверов, и т.п.)

Приемы и способы достижения открытости:

* Применение единых языков определения интерфейсов (IDL)
* Использование единых способов описания общей и частных моделей данных (например, информационная модель б/д)
* Использование принципа отделения правил от механизмов, их реализации
* Использование стандартизованных (как минимум, широко распространенных) решений.

Открытость: за и против

* Достоинства:
  + Использование стандартизованных решений позволяет сократить затраты на разработку
  + Использование общих языков описаний облегчает:
    - Проектирование системы
    - Написание проектной документации
    - Взаимодействие различных групп разработчиков
  + Облегчение последующего развития РИС
* Недостатки:
  + Не существует общего языка описания структур РИС и поведения компонентов в ее составе
  + Усложнение и удорожание проекта

**Задача** 4. Гибкость: виды

* Гибкость может быть следующих видов:
  + Программной
    - Замена программ (например, выпуск новых версий) и перенос их между хостами РИС
  + Аппаратной
    - Изменение числа и характеристик технических средств
* Структурной
  + Изменение структуры РИС (например, переподчинение элементов, разбиение одной РИС на несколько, слияние нескольких РИС в одну)
    - Логической
  + Изменение бизнес процессов (например, последовательности операций)

Гибкость: за и против

* Гибкость – способность приспособления ИС к новым потребностям пользователей.
* Достоинства:
  + Гибкость тесно связана с масштабируемостью
  + Облегчена поддержка РИС в течение долгого периода (например, обновление программ и технических средств)
  + Возможна доработка РИС для решения новых задач
* Недостатки:
  + Увеличение затрат на разработку гибкой РИС
  + Наличие гибкости приводит к расшатыванию устоявшихся приемов работы
  + Дороговизна сопровождения и поддержки
  + Сложность обучения персонала

**Задача** 5. Масштабируемость

Масштабируемость – сохранение работоспособности системы при изменении масштаба системы.

* Масштабируемость системы может измеряться по трем показателям:
  + По размеру.
  + По географической распределенности.
  + По степени сохранения управляемости при увеличении масштаба РИС

Масштабирование по размеру

Наиболее часто проблемы размера определяются заложенными при

ее создании централизацией:

* служб;
* данных;
* алгоритмов.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Масштабируемость и производительность

* Вертикальное масштабирование – увеличение производительности каждого компонента системы с целью повышения общей производительности.
  + –возможность заменять в существующей вычислительной системе компоненты более мощными.
* Горизонтальное масштабирование – возможность добавлять к системе новые узлы (серверы, процессоры) для увеличения общей производительности.

Масштабируемость и устойчивость

* Изменение хостов на более мощные:
  + Синхронизация потоков данных и/или скорости вычислений
  + Обеспечение программно-аппаратной совместимости
* Добавление новых хостов:
  + Увеличение числа связей между элементами (например, по квадратичному закону).
  + Увеличение потока данных, обрабатываемых каждым узлом.
* Изменение (увеличение) нагрузки:
  + Буферизация запросов (в случае неравномерности нагрузки)
  + Приоритетизация обслуживаня запросов.
  + Диагностика перегруженности .
  + Перераспределение нагрузки по узлам.

Технологии масштабирования

Имеется три основные технологии масштабирования:

* Сокрытие времени ожидания выполнения удаленного запроса (времени ожидания связи);
* Распределение компонентов или их частей по различным узлам системы (организация параллельной работы компонент)
* Репликация данных и компонент системы. Кэширование – особая форма репликации.

1. Типы распределенных систем.

Типы распределенных систем

* Системы высокопроизводительных распределенных вычислений (High performance distributed computing).

Высокопроизводительные вычисления возникли с появлением мультипроцессорных и мультимашинных систем.

* Системы распределенных параллельных вычислений (Distributed information systems).
* Повсеместные - всеобъемлющие системы (Pervasive systems). Это мобильные и встраиваемые системы, сюда же относят системы слежения обладающие сетью датчиков и множеством исполнительных механизмов и др.

Всеобъемлющие (всепроникающие) системы (Pervasie systems)

* К этому типу РИС относятся:
* Вездесущие (Ubiquitous systems) системы – пользователи даже не подозревают, что взаимодействуют с ними. Например, системы организации движения в городах, системы видеонаблюдения и т.п.
* Мобильные компьютерные системы.
* Системы сенсоров (датчиков).
* Интернет вещей (Internet of Things – IoT).

Три новые парадигмы распределенных вычислений

* **Сервисы Web 2.0.**

становятся доступными с внедрением **SOA**.

* **Облачные вычисления**.

Достижения в области виртуализации позволяют рассматривать рост интернет-облаков как новую вычислительную парадигму.

* **Интернет вещей** (Internet of things).

Зрелость таких технологий как:

радиочастотной идентификации (RFID),

глобальной системы позиционирования (GPS)

и сенсорных технологий

дала толчок развитию Интернета вещей (IoT).

1. Grid системы. Архитектура GRID. Системы облачных вычислений.

Grid системы

* **Грид – это система, которая координирует распределенные ресурсы** посредством стандартных, открытых, универсальных протоколов и интерфейсов для обеспечения нетривиального ка-чества обслуживания (QoS – Quality of Service).

Ян Фостер книга «Грид. Новая инфраструктура вычислений» 1998 г.

* Концепция грид-вычислений идея подобна концепции электросети (англ. Power Grid): нам не важно, откуда к нам в розетку приходит электричество. Независимо от этого мы можем подключить к электросети утюг, компьютер или стиральную машину.
* Аналогично и в идеологии грид: мы можем запустить любую задачу с любого компьютера или мобильного устройств на вычисление, ресурсы же для этого вычисления должны быть автоматически предоставлены на удаленных высокопроизводительных серверах, независимо от типа нашей задачи.

Архитектура Grid систем

* Ключевой проблемой grid является объединение ресурсов из различных организаций для совместного использования. В результате формируется федерация систем, которая влечет за собой образование виртуальных организаций.
* Для построения Grid была разработана и принята в качестве стандарта *OGSA (Open* Grid Services Architecture – Открытая архитектура грид-сервисов), основанная на архитектуре SOA – Service Oriented Architecture.

Облачные вычисления (Cloud Computing)

* Является развитием концепции Utility Computing – (утилитарный, практичный) предоставление обслуживания запросов по заявка пользователей с оплатой основанной на учете использованных ресурсов.
* Облачные вычисления характеризуются предоставлением пользователю легко доступных виртуализированных ресурсов.

1. Программные системные решения для РИС: распределенные ОС; сетевые ОС; ПО промежуточного слоя (уровня). Краткая характеристика этих решений.

Распределенная ОС

* Это сильносвязанная ОС которая используется для управления мультипроцессорными и гомогенными многомашинными системами.

Сетевые ОС

* Это слабосвязанные ОС предназначенные для управления гетерогенными вычислительными системами.
* Эти системы должны обеспечивать удаленный доступ к локальным службам.

Средства промежуточного уровня

* Их задача обеспечение прозрачности распределения.

Соотношение программных решений РИС и модели OSI



**Сетевые операционные системы**

* В противоположность распределенным операционным системам сетевые операционные системы не нуждаются в том, чтобы аппаратное обеспечение, на котором они функционируют, было гомогенно и управлялось как единая система.
* Напротив, обычно они строятся для набора однопроцессорных систем, каждая из которых имеет собственную операционную систему.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

**Программное обеспечение  
промежуточного уровня**

* Каждая локальная система, составляющая часть базовой сетевой операционной системы, предоставляет управление локальными ресурсами и простейшие коммуникационные средства для связи с другими компьютерами.
* задача промежуточного уровня — скрыть разнообразие базовых платформ от приложений.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

**Модели промежуточного уровня**

* Программное обеспечение промежуточного уровня базируется на некоторой модели, или *парадигме, определяющей* **распределение**и **связь**.
* Например, такой моделью является представление **всех** наблюдаемых **объектов** в виде **файлов**. Этот подход был изначально введен в UNIX.
* Другая важная ранняя модель программного обеспечения промежуточного уровня основана на *удаленных вызовах процедур (****Remote Procedure Calls****), RPC*
* Далее появились системы промежуточного уровня, реализующих представление о *распределенных объектах {****distributed objects****)*
* Успех среды Web в основном определяется тем, что она построена на базе потрясающе простой, но высокоэффективной модели *распределенных документов {****distributed documents****).*
* Затем возникла модель основанная на использовании ***Веб сервисов***

**Службы промежуточного уровня**

* Для систем промежуточного уровня существует некоторое количество стандартных служб. Это:
  + Служба коммуникаций реализующая прозрачность доступа.
  + Служба именования. Проблема состоит в том, что для эффективного поиска имени в большой системе местоположение разыскиваемой сущности должно считаться фиксированным.
  + Служба распределенных транзакций.
  + Служба обеспечения безопасности.

**Промежуточный уровень и открытость**

* Для гарантии совместной работы различных реализаций необходимо, чтобы к сущностям разных систем можно было одинаково обращаться.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

1. Модель клиент-сервер и трехуровневая организация приложений. Варианты архитектуры клиент-серверных систем.

**Архитектура клиент-сервер**



* В классической архитектуре “клиент-сервер” любая информационная система должна иметь минимум три основные функциональные части:
  + модуль хранения данных;
  + модуль обработки;
  + модуль интерфейса с пользователем.

**Модель клиент-сервер**

* В базовой модели клиент-сервер все процессы в распределенных системах делятся на две возможно перекрывающиеся группы. Процессы, реализующие некоторую службу, например службу файловой системы или базы данных, называются *серверами {servers). Процессы, запрашивающие службы у серверов путем посылки* запроса и последующего ожидания ответа от сервера, называются *клиентами {clients).*

Изображение выглядит как линия, текст, Шрифт, чек

Автоматически созданное описание

**Разделение приложений по уровням**

* Рассматривая множество приложений типа клиент-сервер, предназначенных для организации доступа пользователей к базам данных, многие рекомендовали разделять их на три уровня.
  + уровень пользовательского интерфейса;
  + уровень обработки;
  + уровень данных.

**Три основных уровня корпоративных приложений**

* Представление
* Бизнес-логика приложения
* Источник данных



**Варианты архитектуры клиент-сервер**

* Простейшая организация предполагает наличие всего двух типов машин.
  + Клиентские машины, на которых имеются программы, реализующие только пользовательский интерфейс или его часть.
  + Серверы, реализующие все остальное, то есть уровни обработки и данных.
* На самом деле такая система не является распределенной: все происходит на сервере, а клиент представляет собой не что иное, как простой терминал.

**Физически двуззвенные архитектуры**

* Один из подходов к организации клиентов и серверов — это распределение программ, находящихся на уровне приложений, по различным машинам.

Физически трехзвенная архитектура

* Рассматривая только клиенты и серверы, мы упускаем тот момент, что серверу иногда может понадобиться работать в качестве клиента.

**Архитектура унифицированного доступа к информационным ресурсам**



Архитектура многоуровневых клиент-серверных систем достаточно хорошо согласуется с применением современной Web-технологии построения информационных систем. В этом случае в качестве клиентской части используется (одна или с расширениями) унифицированная, единая для всех клиентов программа - просмотрщик (Web-броузер), а сервер приложений дополняется Web-сервером и программами вызова процедур сервера.

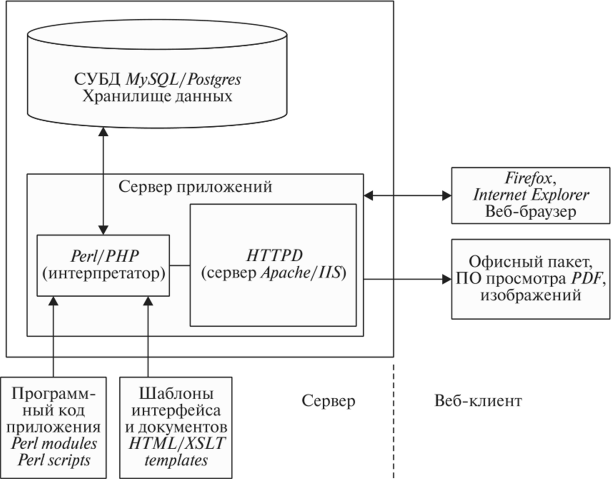
1. Понятие организации распределенной системы. Программная организация. Прозрачность РИС и ее архитектура. Выбор варианта программной архитектуры. Системная архитектура. Виды системной архитектуры.

Понятие ОРГАНИЗАЦИИ распределенной системы

* Организация РС определяется тем каким образом программное обеспечение РС распределяется между вычислительными узлами этой системы.
* В организации систем часто выделяют:
  + Логическую организацию совокупности программных компонент системы;
  + Физическую организацию размещение этих компонент на узлах системы.

Программная ОРГАНИЗАЦИЯ

* Организация РС определяется составом программных компонент входящих в состав системы.
* Программная организация показывает из каких программных компонентов состоит система, а также и то как взаимодействуют между собой программные компоненты этой системы.



Прозачность РиС и ее Архитектура

* Исходя из требования обеспечения прозрачности в распределенных системах требуется четко разделять приложения и лежащие в их основе платформы.
* Такое разделение в РС выполняется с помощью промежуточного уровня системы.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Выбор варианта программной архитектуры

* Важнейшим решением при разработке архитектуры системы является:
  + выбор варианта размещения ПО промежуточного уровня (ППУ) системы.
* Имеется различные методики определение состава и размещения ППУ приложений, что и определяет множество вариантов программных архитектур.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Системная архитектура

* Фактическая (реально разворачиваемая) реализация РС, требует однозначного определения размещения программных компонент системы на реальных машинах.
* Практически всегда имеется множество вариантов такого размещения.
* Размещение программных компонент системы (программная архитектура) на физических машинах называется системной архитектурой.

Изображение выглядит как текст, диаграмма, Параллельный, снимок экрана

Автоматически созданное описание

виды системной архитектуры

* Различают три вида системной архитектуры:
  + централизованная;
  + децентрализованная (peer-to-peer);
  + гибридная – комбинация элементов централизованной и децентрализованной архитектур.

1. Понятие архитектуры (архитектурного стиля) РИС. Понятие программного компонента. Основные виды архитектур РИС.

Понятие архитектурного стиля

* В настоящее время исследования в области программного обеспечения достигли достаточной зрелости, что позволило однозначно определить понятие архитектурного стиля (архитектуры) РИС.
* При проектировании и создании РС выбор архитектуры является ключевым техническим решением, определяющим успех создания больших программных систем.
* При обсуждении архитектурных аспектов РС важным понятием является архитектурный стиль, который описывается в терминологии компонент и определяет:
  + способ коммуникаций между компонентами;
  + порядок обмена данными между компонентами;
  + как элементы системы совместно формируют распределенную систему.

Понятие программного компонента

* **Компонент** – модульная единица ПО снабженная полностью определенным и предоставляемым по запросу интерфейсом.
* Компонент должен обладать свойством – ***заменяемости*** *(replaceable)* в рамках системного окружения. Замена компонента может быть выполнена в любой момент, даже в условиях работы системы. Последний аспект определяет, что в РС может отсутствовать опция
* **Интерфейс** - описывает состав параметров необходимых для обращению к компоненту. Замена компонента может быть выполнена только при условии неизменности его интерфейса.
* **Конектор** – это механизм который обеспечивает коммуникации, и способствует координации (или кооперации) компонент друг с другом.
* Конектор может быть сформирован на основе средств реализующих способ связи между компонентами. :
  + удаленный вызов процедур (RPC);
  + обмен сообщениями (message passing);
  + потока данных (streaming data) и д.р.
* Использование понятий компонент и конектор позволяет описывать различные варианты конфигураций, которые в свою очередь могут быть квалифицированы как **архитектурные стили**.

Основные виды архитектур РИС

* В настоящее время общепризнанными архитектурными решениями (стилями) считаются:
  + многоуровневые архитектуры (layered);
  + объектные архитектуры (object-based);
  + компонентные архитектуры (component-based);
  + сервисно-ориентированные архитектуры;
  + ресурсо-центрированные архитектуры (resource-centerd) ;
  + архитектуры основанные на событиях (event-based).
* Эти архитектурные решения могут одновременно применяться в одних и тех же системах в различных сочетаниях.

1. Архитектурные элементы как сущности распределенной ИС. Системный и проблемный аспекты выбора сущностей архитектуры РИС. Объекты, объектные компоненты, сервисы и ресурсы – как сущности архитектуры.

Архитектурные элементы

* Для понимания того, что является строительными блоками распределенных систем, необходимо решить четыре ключевых вопроса:
  + 1. Какие сущности взаимодействуют в рамках распределенной системы?
  + 2. Каким образом эти сущности взаимодействуют друг с другом? Или более точно: какую коммуникационную модель (парадигму) они используют?
  + 3. Какие роли играют (выполняемые ими функции), и какую ответственность они несут в рамках всей системы?
  + 4. Каким образом сущности системы (логические) отображаются на физические элементы распределенной инфраструктуры?

Сущности РИС

* АРХИТЕКТУРА РИС определяет:
  + какие сущности входят в РС,
  + каким образом они взаимодействуют.
* Выбор сущностей и способы их взаимодействия в рамках РИС определяет весь спектр возможных архитектурных решений.
* Сущности, образующие РИС, имеют два аспекта:
  + системный;
  + проблемный.

Системная реализация сущности

* Системная сторона сущности определяет способ **физической реализации функций сущности** и методов ее **взаимодействия** с другими сущностями.
* В современных ИС способ такой физической реализации является **процесс** (**задача**) ОС, выполняемый на аппаратных средствах физического узла и получает для своего исполнения все необходимые ресурсы (вычислительные, ОП, устройства хранения (SDD, HDD), ВУ, сетевые устройства).
* Исходя из системной стороны сущностей, взаимодействие между ними рассматривается как модель **межпроцессного** взаимодействия между распределенными сущностями.
* При этом помимо классического процесса имеются следующие варианты их реализации:
  + В примитивных системах, например, таких как системы датчиков, ОС узлов системы могут не поддерживать механизм процессов, в этом случае такие сущности называют **узлами**;
  + В большинстве современных ОС абстракция процесса дополняется абстракцией **потока**. Потоки, как и процессы могут обслуживать **конечные точки подключений**.
* Точка подключения: - сетевой адрес узла/процесса/потока. Пример: URL/URI

Проблемная сторона сущности

* Определяется предметной областью, для автоматизации которой используются сущности РС. Предметная сторона определяет:
  + алгоритмы обработки информации;
  + порядок обмена информацией между сущностями в рамках РС.
* Наиболее часто в качестве проблемно-ориентированной абстракции сущностей РИС используются:
  + объекты;
  + компоненты;
  + службы;
  + ресурсы.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Объекты

* Выбор объектов в качестве сущностей РИС обусловлен во многом объектно-ориентированным подходом в программировании, а также к проектированию РИС.
* Объекты РС представляют собой абстракцию объектов реального мира, относящихся к конкретной проблемной области.
* Разделение предметной области на объекты выполняется исходя из естественной декомпозиции предметной области на ряд взаимодействующих друг с другом объектов.
* Доступ к объектам реализуется через их интерфейсы, которые описываются с помощью языка IDL-Interface Definition Language, обеспечивающего описание методов, определенных для объекта.
* Распределенные объекты наиболее часто используются в качестве проблемных сущностей в распределенных системах.

Объектные компоненты

* Использование объектов в качестве сущностей РС привело к появлению ряда проблем, ответом на которые явилось использование компонент. Компоненты сходны с объектами. Они также как объекты:
  + являются абстракциями прикладной области;
  + доступны через интерфейсы.
* Ключевым отличием компонент от объектов состоит в том, что они не только предоставляют интерфейсы, но также содержат информацию о том, какие компоненты/интерфейсы должны быть представлены для компонента, чтобы он мог выполнять свои функции.
* Иными словами, явно разрешаются все зависимости, и обеспечивается полное выполнение всех правил, принятых при конструировании системы, что позволяет разработку компонент третьими сторонами и способствует при реализации проекта более полной и адекватной реализации.

Сервисы

* Сервисы представляют третью парадигму определения сущностей, используемую при создании РС.
* Сервисы являются сущностями близкими к объектам и компонентам, также использующими подход, основанный на инкапсуляции поведения в компонент и использовании интерфейса для получения доступа друг к другу.

Web-сервисы

* В отличие от первых двух видов сущностей web-сервисы в действительности интегрированы в web и используют стандарты www для представления и описания сервисов.
* W3C определил Web-сервисы как:
  + Программное приложение, идентифицируемое с помощью URI, чей интерфейс и связывание реализуется на основе использования XML.
  + Web-сервис поддерживает прямое взаимодействие с другими программными агентами, используя обмен XML-сообщениями по Internet протоколам.
* Если объекты и компоненты часто используются внутри организаций для разработки тесно связанных приложений, то Web-сервисы в целом выглядят как полноценные сервисы, которые могут при их комбинировании образовывать сложные сервисы с дополнительными возможностями.
* Web-сервисы часто используются для интеграции B2B, что требует от них преодоления границ, разделяющих организации. Web-интерфейсы могут быть реализованы разными провайдерами и использовать различные низкоуровневые технологии.

Ресурсы

* Рост числа сервисов доступных через Web и создание распределенных систем на основе композиций сервисов привели к необходимости переосмысления архитектуры РИС построенных на базе Web.
* Одной из проблем построения РИС на основанных на Web сервисах явилась высокая сложность обеспечения связи между большим числом различных компонент.
* В качестве альтернативы было предложено рассматривать РИС как коллекцию ресурсов каждый из которых индивидуально управляется своим компонентом (сервисом).

1. Модели коммуникаций при взаимодействии между сущностями архитектуры РИС: межпроцессные коммуникации (IPC – Interprocess communications). Удаленный вызов. Непрямые коммуникации (indirect communications). Групповые коммуникации.

Модель взаимодействия распределенной системы

* В отличие от локальных систем для которых понятие алгоритма работы программы знакомо всем программистам, в распределенных системах поведение системы определяется **взаимодействием процессов** работающих на удаленных машинах друг с другом и описывается **распределенными алгоритмами работы системы**.
* Поведение и состояние распределенной системы определяется теми **действиями которые выполняются на удаленных** узлах, а также **сообщениями**, которыми они обмениваются друг с другом.
* **Взаимодействие** между процессами определяет всю **функциональность распределенной системы**. При этом каждый процесс имеет свое состояние, обладает своим набором данных, включая переменные которое определяется узлами

Модели коммуникаций

* Имеется три вида коммуникаций между сущностями РС:
  + Межпроцессные коммуникации (прямые коммуникации);
  + Удаленный вызов компонент;
  + Непрямые коммуникации.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Межпроцессные (прямые) коммуникации (IPC – Interprocess communications)

* Этот вид коммуникаций относится к низкоуровневому способу поддержки взаимодействия **между процессами** выполняющимися различными сущностями в распределенной системе (на узлах).
* К этому типу коммуникаций относятся следующие типы:
  + Обмен сообщениями (message passing);
  + Прямой доступ к API протоколов Internet (программирование сокетов TCP/IP);
  + Групповые коммуникации на основе групповой адресации (multicast addresses).

Удаленный вызов

* Представляет собой способ коммуникаций **наиболее** **широко используемый** в распределенных системах.
* Объединяет ряд методов основанных на двустороннем обмене информацией между сущностями:
  + Запрос-ответ.
  + Вызов удаленных процедур (RPC – Remote Procedure Call).
  + Удаленный вызов методов (RMI – Remote Method Invocation).

непрямые коммуникации (indirect communications)

* К непрямым коммуникациям относятся способы взаимодействия между двумя сущностями посредством третьей стороны, что **уменьшает степень связности** между передатчиком и приемником. Такие варианты взаимодействия возникают, например когда:
  + передающей стороне не требуется знать, кто является получателем (**пространственное разъединение**);
  + передающей и принимающей сторонам не требуется находиться в рабочем состоянии в одно и то же время (**временное разъединение**).

Модели непрямых коммуникаций

* К этому виду моделей относятся:
  + групповые коммуникации (group communication);
  + системы публикации-подписки (publish-subscribe systems);
  + очереди сообщений (message queues);
  + кортежное пространство (tuple spaces);
  + распределенная разделяемая память (distributed shared memory).

Групповые коммуникации

* Эта модель предполагает доставку сообщений набору получателей. Соответствует парадигме один-ко-многим. Групповая модель взаимодействия основана на абстракции группы, представленной в системе идентификатором группы.
* Получатель подключается к группе путем отправки сообщения о присоединении к группе. Отправители отправляют сообщения в группу по ее идентификатору и не знают конкретный состав получателей.
* Группы, как правило, обслуживают членство в группе, для чего имеют механизм выхода из группы.

1. Модели коммуникаций при взаимодействии между сущностями архитектуры РИС: системы публикации-подписки (publish-subscribe). Очередь сообщений (Message queues). Пространство записей (кортежей) (Tuple spaces). Распределенная разделяемая память (Distributed Shared Memory).

Системы публикации-подписки (publish-subscribe)

* Примером таких систем могут служить системы торговли различными ресурсами, в которых большое число производителей (публикаторов) распространяют информацию среди большого числа потребителей (подписчиков), для таких систем не подходят модели прямых коммуникаций.
* Системы публикации-подписки содержат промежуточный сервис, который является ключевым элементом этих систем.
* Этот сервис должен обеспечивать гарантированное распространение информации, генерируемой публикаторами ко всем подписчикам, желающим получать данную информацию.

Очередь сообщений (Message queues)

* В то время как системы публикации-подписки поддерживают стиль один-ко-многим, очереди сообщений предлагают модель обслуживания один-к-одному (точка-точка).
* В этом случае процесс-производитель сообщений отправляет их в определенную очередь, а процесс-потребитель может получать сообщения из этой же очереди либо получать уведомления о поступлении в очередь новых сообщений.

Пространство записей (кортежей) (Tuple spaces)

* Пространство кортежей (Tuple spaces) предполагает, что процесс-отправитель может поместить некоторый произвольный экземпляр структурированных данных (кортеж) в постоянное пространство записей, а другие процессы могут либо читать, либо удалять такие записи с помощью шаблонов, определяющих интересующую их информацию.
* Так как пространство записей является постоянным, процессы "читатели" и процессы "писатели" не должны существовать в одно и то же время.
* Этот стиль программирования также известен как "генерирующие" коммуникации, был предложен Джелернтером (Gelernter) как модель для параллельного программирования.
* Имеется ряд реализаций этой модели коммуникаций, адаптированных как к системам клиент-сервер, так и для распределенных систем Р2Р.

Распределенная разделяемая память (Distributed Shared Memory)

* Распределенная разделяемая память (Distributed Shared Memory) представляет собой абстракцию разделения данных между процессами, которые не разделяют между собой физическую память.
* Тем не менее, программисты получают возможность писать/читать структуры данных также легко, как если бы данные находились в собственном адресном пространстве процесса.
* Ниже лежащая инфраструктура должна гарантировать выполнение своевременного копирования и синхронизации данных DSM.

1. Роли и ответственность сущностей РС. Архитектура клиент-сервер. Архитектура Peer-to-Peer.

Роли и ответственность сущностей РС

* В распределенной системе процессы, представленные соответстветствующими объектами, компонентами или сервисами (включая Web-сервисы), **взаимодействуют друг с другом**, выполняя необходимые действия, в рамках выполняемых ими функций (ролей), например, поддержка сессии чата.
* Роль, выполняемая сущностью, является **фундаменто**м для определения архитектуры системы.
* Имеется два вида архитектурных систем, определенных **на основе ролей**, выполняемых элементами РС:
  + клиент-сервер;
  + peer-to-peer.

Клиент-сервер

Эта архитектура получила очень широкое распространение, благодаря свой простоте и широкому распространению в сети Интернет.

На клиенте и сервере используется разное ПО.

Архитектура Peer-to-Peer

* В этой архитектуре все процессы участвуют в работе на основе **одной общей для всех роли**, которая не накладывает никаких ограничений, связанных с выполнением ролей клиента или сервера.
* Все сущности используют **одно и тоже программное обеспечение** и предоставляют один и тот же интерфейс для взаимодействия с другими сущностями.

1. Системная архитектура РИС. Стратегии размещение логических сущностей на элементах физической инфраструктуры РИС.

Стратегии Размещения сущностей

* Определяет каким образом **логические сущности** системы соотносятся с **физической инфраструктурой** системы, которая может содержать большое число машин связанных между собой сетью произвольной сложности.
* Размещение критически влияет на такие свойства распределенных систем как:
  + Производительность;
  + Надежность;
  + Безопасность.
* Вопрос размещения ПО заданного клиента или сервера на конкретных машинах, а также определение процессов выполняемых в каждой из машин распределенной системы является **предметом разработки архитектуры** РС. Размещение требует знания требований к размещению со стороны приложений.
* Имеется несколько хорошо известных **стратегий размещения**:
  + Отображение служб на несколько серверов (mapping of services to multiple servers);
  + Кэширование;
  + Использование мобильного кода;
  + Использование мобильных программных агентов.

Мультисерверная архитектура сервисов

* Службы могут быть реализованы как несколько серверных процессов выполняемых в рамках одного или нескольких физических серверов. Примеры:
  + Каждый Веб сервер управляет своими собственными ресурсами. Браузер может премещаться между серверами получая доступ к ресурсам каждого из них.
  + При обработке запросов пользователей в системе построенной на основе сервисов, при обработке запроса клиента один сервис может обращаться к другим сервисам для обслуживания. В этом случае говорят о сложных сервисах.

Кэширование

* Кэш – это хранилище последних данных полученных в результате взаимодействия между сущностями РС. При очередном обращении к сущности полученные в результате данные добавляются в это хранилище.
* Ярким примером использования кэширования является кэширующий веб прокси сервер.

Мобильный код

* Примером мобильного кода являются JavaScript (VBScript), Java апплеты.
* Использование мобильного кода позволяет перенести выполнение действий с сервера на клиент.
* Позволяет улучшить время реакции, но требует дополнительных ресурсов со стороны клиента.

Мобильные программные агенты

* Мобильный программный агент – это исполняемая программа (код + данные), которая перемещается между машинами распределенной системы по сети и в конечном итоге возвращается с результатом.
* Мобильный агент может выполнять много обращений к локальным ресурсам машины которую он посещает.
* По сравнению с удаленными обращениями к ресурсам применение мобильных агентов позволяет уменьшить стоимость коммуникаций и время получения нужных данных.
* Мобильные агенты могут использоваться для:
  + Установи и обслуживания ПО на компьютерах внутри организации.
  + Сравнения цен от ряда поставщиков, путем посещения их сайтов.
  + Выполнения серии операций с базами данных, размещенных на разных машинах.
  + Использовать время простоя машины для выполнения распределенных вычислений.
* Использование мобильных агентов таит в себе опасности нарушения безопасности ресурсов машин которые они посещают.

1. Многоуровневая архитектура РИС. Типовая архитектура корпоративного приложения. Эволюция многоуровневой архитектуры.

Многоуровневая архитектура

* Базовая идея проста: компоненты распределяются по уровням, при этом компонент уровня j Lj может вызывать компонент находящийся на нижележащем уровне i (Li), от которого он получает ответ.
* Как правило вызов компонента выше стоящего уровня нижележащим запрещен.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Другие виды МНОГОУРОВНЕВЫХ АРХИТЕКТУР

* Смешанная многоуровневая архитектура – допускает вызов не только смежных нижележащих уровней, но и более глубоколежащих уровней.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Многоуровневая организация стека сетевых протоколов

* Примером такой организации может служить стек сетевых протоколов, например : TCP/IP

Изображение выглядит как текст, диаграмма, число, снимок экрана

Автоматически созданное описание

**Архитектура приложения  
(расслоение приложения)**



Многоуровневая организация приложений Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Эволюция многоуровневой архитектуры

1960-1970. Терминальные приложения

* Терминальные приложения 1960-1970-е были по сути одноуровневыми.

1980-1990. Корпоративные приложения

* С появлением технологий б/д сформировались три классических уровня

Середина 1990-х.  
Корпоративные Web-приложения

* Уровень приложения разделился на подуровни:
  + Представления;
  + Сервер приложений;
  + Модель данных;
  + Перманентность объектов.

Начало 2000-х. архитектура систем управляемых бизнес логикой

* UI – уровень взаимодействия с пользователем, где пользователем может быть не только человек, но и программа.
* AL – уровень приложения, выполняет координацию (оркестровку) объектов предметной области, относящихся к уровню приложения.
* DL – уровень бизнес логики, сущностей, событий и других объектов. **Является сердцем системы**.
* Infrastructure - поддержка перманентных объектов и обмен сообщениями.

1. Трехзвенная архитектура Веб-приложения. Методы размещения функциональности на стороне сервера. Достоинства и недостатки.

Расширение функциональности на стороне сервера

* Использование CGI
* FastCGI
* API-интерфейсы
* SSI–расширения функциональности на стороне сервера с помощью использования служебных тэгов
* Использование серверов приложений

**Использование CGI**

* CGI был первым протоколом, позволившим разработчикам писать программы, наращивающие функциональность Web-серверов.
* Большинство ранних баз данных для WWW были написаны с использованием CGI, и обычная архитектура с CGI до сих пор обладает наибольшей переносимостью между различными Web-серверами. Обычный CGI можно найти и во многих специализированных приложениях баз данных для WWW, частично по причине доступности бесплатных CGI-процедур.

**Недостатки CGI**

* поскольку на сервере для каждого очередного запроса порождается новый процесс, который завершается после его выполнения, то это приводит к невысокому быстродействию CGI-скрипта и снижает эффективность работы сервера.
* при использовании CGI-программ для доступа к базам данных из-за отсутствия поддержки непрерывного соединения Web-сервера и соответствующей СУБД очень сложно обеспечить процесс “ведения” пользователя базой данных, т.к. каждый раз при генерации очередного запроса требуется новое подключение

**Использование гибридного CGI (fast CGI)**

* При каждом запросе от броузера Web-сервер вызывает маленькую CGI-программу и передает ей данные. Она же просто передает данные процессу-партнеру, почти ничего не делая.
* Этот процесс-партнер (системный сервис в Windows NT или демон в Unix) загружается только раз, обычно при загрузке операционной системы, и остается работать в **фоновом режиме**.
* Практически всю полезную работу выполняет процесс-партнер. Поэтому CGI-программа может быть очень маленькой, загружаться гораздо быстрее и отнимать намного меньше ресурсов. Процесс-партнер может улучшить быстродействие еще и за счет сохранения соединения с базой данных после завершения CGI-программы.

**Гибридный (fast) CGI**

**API-интерфейсы**

* В ответ на ограничения и недостатки спецификации CGI была разработана спецификация прикладных модулей API, встроенных в сервер. Данное расширение Web-сервера запускается как динамическая библиотека и выполняет обработку каждого вызова сервера по отдельной структуре памяти, что значительно проще, чем создание отдельного процесса для каждого клиентского запроса.

**Сервер приложений**

* В архитектуре клиент-сервер функции, реализующие взаимодействие с сервером баз данных возлагаются на сервер приложений. Естественным решением при организации взаимодействия сервера WWW с базами данных может быть объединение функций WWW-сервера и сервера приложений в один программный продукт.

1. Трехзвенная архитектура Веб-приложения. Методы размещения функциональности на стороне клиента. Достоинства и недостатки.

**Расширение функциональности на стороне клиента**

* Вспомогательные программы (Helpers)
* Подключаемые модули (Plug-ins)
* Java-апплеты
* Технология Axtive X
* Использование языков описания сценариев (JavaScript, Vbscript)

**Вспомогательные программы (Helpers)**

* Вспомогательная программа представляет собой отдельную программу, имеющуюся на персональном компьютере пользователя и вызываемую броузером. Чтобы такая схема работала, необходимо предварительно сконфигурировать броузер таким образом, чтобы он запускал определенную вспомогательную программу, когда пользователь щелкает мышью по ссылке, содержащей определенное расширение имени файла.
* Особенностью вспомогательных программ является то, что броузер скачивает данные во временный файл перед тем, как вызвать вспомогательную программу. В случае с большими файлами это может занимать много времени.

**Подключаемые модули (Plug-ins)**

* Подключаемые модули схожи с вспомогательными программами в том, что в их задачи входит обработка и отображение данных, которые броузер не в состоянии обработать самостоятельно. Однако подключаемые модули более тесно интегрированы с броузером. Их также можно запрограммировать таким образом, чтобы они начинали показывать файл еще до того, как он будет полностью передан.
* Подключаемые модули существуют в двух разновидностях: подключаемые модули для Netscape и компоненты ActiveX фирмы Microsoft. Оба вида требуют от пользователя заранее установить программу на свой компьютер.

**Java-апплеты**

* Java-апплеты представляют собой откомпилированные программы, загружаемые после запроса на просмотр HTML-страницы и затем запускаемые броузером.
* Апплеты работают как программы, написанные на языке интерпретатора, что снижает вероятность передачи через них вируса, поскольку каждый оператор перед выполнением проверяется.
* Java-апплеты являются кросс-платформенным решением, что делает их очень мощным средством.
* Одно из серьезных преимуществ Java состоит в возможности управления версиями - поскольку апплет передается на компьютер пользователя каждый раз заново, всегда можно быть уверенным, что будет запущена самая последняя версия.

**Схема взаимодействия Java-приложений с сервером БД**

****

**Использование языков описания сценариев**

* Сценарии - это программы, внедренные в тело HTML-страницы. Сценарии отлично интегрируются с броузером, поскольку они добавляют функциональные возможности без изменения обычного вида Web-страницы.
* Язык JavaScript, разработан фирмами Netscape и Sun.
* Microsoft энергично пропагандирует VBScript и свой вариант JavaScript, который получил название Jscript.
* В любом случае сценарии отлично вписываются в системы баз данных для WWW ввиду своей способности предоставлять средства проверки пользовательского ввода в HTML-формы.

1. Архитектурные шаблоны построения многозвенных приложений: - шаблон многозвенной архитектуры с тонким клиентом; шаблон с использованием прокси-сервера; шаблон с использованием брокеров; шаблон с использованием рефлексии.

Архитектурные шаблоны

* Архитектурные шаблона основаны на совместном использовании примитивных архитектурных элементов (сущностей, типов коммуникаций, ролей, размещений и др.).
* Имеется множество архитектурных шаблонов среди которых, наиболее часто используемыми являются:
  + Шаблон многоуровневой архитектуры приложений;
  + Шаблон многозвенная архитектуры распределенных приложений;
  + Шаблон многозвенной архитектуры с тонким клиентом;
  + Шаблон использования прокси-сервера;
  + Шаблон на основе использования брокеров;
  + Шаблон использования рефлексии.

Многоярусная (многозвенная) архитектура

* Эта архитектура является архитектурой дополняющей многоуровневую архитектуру.
* В то время как многоуровневая архитектура относиться к вертикальной организации сервисов в рамках абстракции уровней, ярусная архитектура является способом организации данного уровня и размещения этой функциональности на серверах, соответствующих физических машин.

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Тонкий клиент

* В распределенных системах наметилась тенденция переноса сложности с клиента на сервер. Это наиболее характерно для облачных вычислений, однако подобная тенденция наблюдается и для многозвенной архитектуры. Это вызвало интерес к концепции тонкого клиента.
* Термин тонкий клиент означает, что уровень пользовательского интерфейса располагается на клиенте, а исполняемый код, реализующий основную функциональность приложения, располагается на сервере (удаленном компьютере).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

* Достоинством такого подхода является возможность использование в качестве клиента простых локальных устройств, например, смартфонов и других устройств с ограниченными ресурсами.
* Этот шаблон характерен для облачных вычислений.

Шаблон Proxy

* Этот шаблон часто используется в распределенных системах, разрабатываемых для локальной прозрачности при использовании RPC или RMI коммуникаций.
* В этом шаблоне прокси создается в локальном адресном пространстве для представления удаленного объекта. Этот посредник предоставляет в точности тот же интерфейс, что и удаленный объект.
* Использование шаблона на основе посредника обеспечивает поддержку локальной прозрачности в RMI и RPC коммуникациях.
* Прокси могут использоваться также для инкапсуляции других видов функциональности, например:
  + Размещения политик;
  + Обеспечения репликаций;
  + Выполнения кэширования.

Использование брокера

* В случае использования в качестве сущности сервиса часто используется архитектура, обеспечивающая взаимодействие между сущностями с помощью брокера сервиса.

Изображение выглядит как линия, белый, снимок экрана, круг

Автоматически созданное описание

Этот шаблон, например, применяется для взаимодействия с реестром сервисов в Java RMI, а также для взаимодействия с сервисом имен в CORBA.

Рефлексия

* Эта модель (шаблон) все чаще используется в распределенных системах как средство обеспечения:

1. Динамического обнаружения свойств системы (интроспекция);

2. Реализация способности динамического изменения ее структуры или поведения (интерцессия).

* Например, способность Java динамически обнаруживать свойства объекта (компонента) эффективно используются в реализации общего диспетчирования в RMI.
* В рефлексивных системах на базовом уровне доступен стандартный интерфейс сервиса, а на мета-уровне становятся доступными компоненты и их параметры, задействованные (участвующие) в реализации сервиса.
* Рефлексия широко используется в РС, особенно при создании ПО промежуточного слоя для реализации реконфигурируемых архитектур.

1. Архитектура, основанная на ресурсах. Ресурсо-центрическая распределенная система. Web-сервисы RESTful.

**Web-сервисы RESTful**

* Технология REST не привлекла большого внимания в 2000 году, когда Рой Филдинг (Roy Fielding) впервые представил ее в Калифорнийском университете в Ирвайне в своей диссертации "Архитектурные стили и дизайн сетевых архитектур программного обеспечения”
* В 2004 году W3C выпустил определение ещё одного стандарта под названием RESTful.
* За последние несколько лет REST стала преобладающей моделью проектирования Web-сервисов. Фактически REST оказала настолько большое влияние на Web, что практически вытеснила дизайн интерфейса, основанный на SOAP и WSDL, из-за значительного более простого стиля проектирования.
* В последние годы этот стандарт стал довольно популярным. На данный момент он используется многими известными сайтами по всему миру, всеми публичными облачными системами.

1. Принципы Архитектуры RESTful. Явное использование HTTP-методов REST сервисами. Web сервис с сохранением и без сохранения состояния. Структура URI REST сервиса. Форматы данных, используемые web-сервисами REST. Сравнение REST и SOAP/XML Web-сервисов.

Архитектура REST основана на четырех принципах:

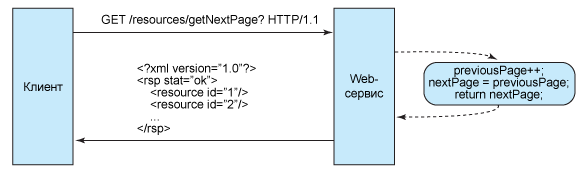
* Идентификация ресурса посредством URI – Ресурсы идентифицируются через URI, который предоставляет глобальное адресное пространство для поиска ресурсов и сервисов.
* Унифицированный интерфейс – GET извлекает текущее состояние ресурса в некотором представлении. POST передает новое состояние ресурса.
* Информативные сообщения – Ресурсы отделены от их представления таким образом, что их содержимое может быть доступно в различных форматах (например, HTML, XML, текст, RDF, JPEG).
* Взаимодействия через гиперссылки – Для обмена существуют различные технологии (например, переименование URI, cookies и скрытые поля формы). Состояние может быть вставлено в ответное сообщение, чтобы указать допустимое будущее состояние взаимодействия.

**Явное использование HTTP-методов**

* Этот основной принцип проектирования REST устанавливает однозначное соответствие между операциями create, read, update и delete (CRUD) и HTTP-методами.
* Согласно этому соответствию:
  + Для создания ресурса на сервере используется POST.
  + Для извлечения ресурса используется GET.
  + Для изменения состояния ресурса или его обновления используется PUT.
  + Для удаления ресурса используется DELETE.

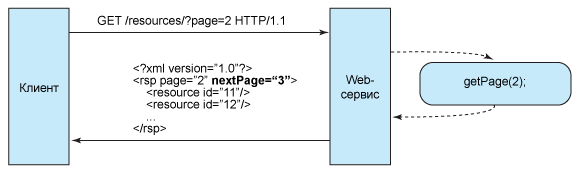
**Web сервис с сохранением состояния**

* На рисунке показан сохраняющий состояние сервис, в котором приложение может запросить следующую страницу в многостраничном наборе результатов, полагая, что сервер хранит последовательность переходов приложения по этому набору результатов.
* В этой модели с сохранением состояния сервис наращивает и сохраняет переменную previousPage



**Web-сервис без сохранения состояния**

* В Web-сервисе RESTful сервер отвечает за генерирование ответов и за предоставление интерфейса, позволяющего клиентскому приложению самому хранить свое состояние. Например, при запросе многостраничного набора результатов клиентское приложение должно включать в запрос номер конкретной страницы, а не просто запрашивать *следующую* страницу (см. рисунок 2).

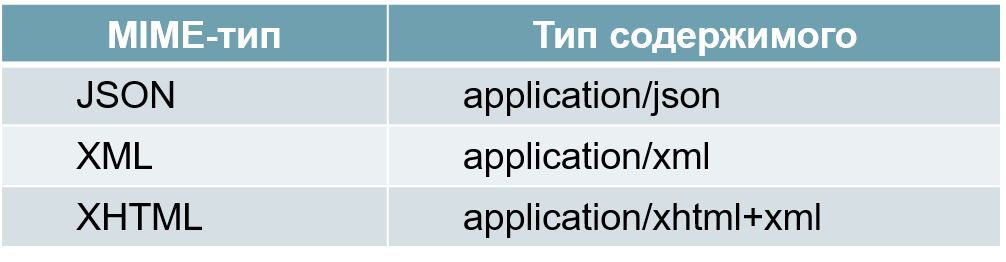


**Структура URI аналогичная структуре каталогов.**

* Структура URI должна быть простой, предсказуемой и понятной.
* Один из способов достичь такого уровня удобства использования – построение URI по аналогии со структурой каталогов. Такого рода URI являются иерархическими, исходящим из одного корневого пути, ветвления которого отображают основные функции сервиса.
* Например, в сервисе обсуждений различных тем:
  + [http://www.myservice.org/discussion/topics/{topic}](http://www.myservice.org/discussion/topics/%7btopic%7d)
  + [http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}](http://www.myservice.org/discussion/%7byear%7d/%7bday%7d/%7bmonth%7d/%7btopic%7d)
  + Дополнительные рекомендации, при обдумывании структуры URI для Web-сервисов RESTful:
* Скрывайте расширения файлов серверных сценариев (.jsp, .php, .asp), чтобы можно было выполнить портирование приложений на другую технологию без изменения URI.
* Используйте только строчные буквы.
* Заменяйте пробелы дефисами или знаками подчеркивания (чем-то одним).
* Старайтесь максимально избегать использования строк запросов.
* Вместо использования кода 404 Not Found для URI, указывающих неполный путь, всегда предоставляйте в качестве ответа ресурс или страницу по умолчанию.

**Передача XML, JSON или обоих**

* Это ограничение, тесно связанный с дизайном Web-сервисов RESTful, относится к формату данных, которыми обмениваются приложение и сервис при работе в режиме запрос/ответ или в теле HTTP-запроса. Здесь особенно важны простота, читабельность и связанность.
* **Общеупотребительные MIME-типы, используемые RESTful-сервисами**



1. Архитектуры, основанные на публикации и подписке (п/п). Способы координации в распределенной ИС. Варианты архитектур публикация/подписка. Основная проблема систем публикации/подписки.

Архитектуры основанные на Публикации и подписке

* По мере роста размеров РС стало важным иметь архитектуру в которой зависимость между процессами стала бы как можно меньше.
* В качестве таковой была предложена архитектура в которой было введено строгое разграничение между процессами передачи и обработки сообщений и координацией этих процессов.
* Идея состояла в том, что бы взглянуть на РС как на совокупность процессов обработки информации взаимодействующих между собой.
* В этой модели координация заключается в коммуникации и координации между процессами. Координация играет роль клея связывающего активность процессов в единое целое.

Способы координации

* В зависимости то вида координации используемой в системе можно определить несколько моделей архитектуры подписка/публикация.
* В качестве способов координации используются:
  + Временн**а**я координация;
  + Ссылочная (адресная) координация.

ВАрианты архитектур публикация/подписка

* В зависимости от комбинации двух факторов координации различают следующие модели архитектуры публикация/подписка:
  + архитектура П/П с прямой координацией;
  + архитектура П/П с координацией через почтовый ящик;
  + архитектура П/П с координацией на основе событий;

архитектура П/П с координацией на основе разделяемых данных.

1. Архитектура систем публикации/подписки с прямой координацией.

Архитектура п/п с прямой координацией

* В этой модели одновременно используются оба вида координации:
  + Координация по времени существует, когда оба процесса запущены и работают).
  + Координация по ссылке (адресная) существует, когда имеется явная ссылка на процесс с которым требуется взаимодействие, либо в виде имени либо в виде его идентификатора.

Изображение выглядит как текст, снимок экрана, диаграмма, мультфильм

Автоматически созданное описание

1. Архитектура систем публикации/подписки с координацией через почтовый ящик.

Архитектура П/П с координацией через почтовый ящик

* В этой модели:
  + Отсутствует координация по времени, оба процесса запускаются и работают не синхронизируясь друг с другом.
    - Подписчик периодически проверяет наличие сообщений в своем почтовом ящике.
    - Отправитель (издатель) периодически активизируется для отправки сообщения в почтовый ящик другого процесса.
  + Координация по ссылке (адресная) существует, в виде адреса почтового ящика.

Изображение выглядит как текст, снимок экрана, мультфильм

Автоматически созданное описание

1. Архитектура систем публикации/подписки на основе событий. Варианты Архитектуры п/п на основе событий. Принцип обмена данными между издателем и подписчиком в системах с шиной событий.

Архитектура п/п на основе событий

* Для получения уведомления процесс подписчик должен всегда находиться в рабочем состоянии.
* Отсутствие ссылочной информации не позволяет процессам иметь явные знания друг о друге.
* Такая архитектура соответствует коммуникациям через общую шину сообщений.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Варианты Архитектуры п/п на основе событий

* В зависимости от способа описания события различают два варианта систем с архитектурой публикации/подписке на основе событий:
  + системы публикации/подписки основанные на теме (topic-based public-subscribe systems);
  + системы публикации/подписки основанные на содержимом (контенте) (content-based public-subscribe systems)

Принцип обмена данными между издателем и подписчиком в системах с шиной событий

* Условием обмена данными является совпадение подписки на событие и уведомление о событии.
* Во многих случаях событие на самом деле соответствует данным, ставшим доступными. В этом случае при совпадении имеется два сценария:
  + Программное обеспечение промежуточного уровня (ПУ) может решить направить подписчикам уведомления вместе с ассоциированными с этим событием данными. Это называется **процессом с совпадением подписки**.
  + Программное **ПУ передает только уведомление**. При этом подписчики смогут самостоятельно считать требуемые данные. ПО ПУ не предоставляет услуг по хранению данных. Услуги хранения оказываются отдельным сервисом.

1. Сравнение систем P2P и клиент-сервер. Сравнение принципов построения систем P2P и клиент-сервер. Способы реализации распределенности в системе. Задачи, решаемые с помощью Р2Р систем. Принципы построения распределенных систем Р2Р.

Сравнение систем P2P и клиент-сервер

Изображение выглядит как диаграмма, линия, дизайн

Автоматически созданное описание

В отличие от классической клиент-серверной архитектуры в архитектуре Р2Р каждый узел, входящий в РС, может играть роль и клиента и сервера. Предоставляя свои либо используя чужие ресурсы.

Можно выделить следующие проблемы архитектуры клиент-сервер:

1. Проблемы масштабируемости.
2. Проблема высокой связности узлов.

Р2Р системы обладают следующими преимуществами:

1. Слабая зависимость от централизованных сервисов и ресурсов.
2. Система допускает сильные изменения в структуре, сохраняя при этом свою работоспособность.
3. Высокая масштабируемость

Сравнение принципов построения систем P2P и клиент-сервер

Сравниваемые принципы организации систем:

* управление;
* доступность ресурсов;
* структура;
* мобильность ресурсов;
* время жизни;
* именование;
* синхронизация процессов.

Изображение выглядит как текст, круг, Шрифт, число

Автоматически созданное описание

Четкой границы между централизованной и децентрализованной архитектурами не существует.

Способы реализации распределенности в системе

* Различают два способа реализации распределенности:
  + Вертикальная распределенность.
  + Горизонтальная распределенность.

Вертикальная распределенность

* Реализуется путем разбиения приложения на логические уровни связанные иерархически.
* Такая организация характерна для **многозвенных клиент-серверных архитектур**.
* Каждый логический уровень размещается на отдельном узле РС
* Термин вертикальная распределенность связан с концепцией вертикальной фрагментации, используемой **в распределенных реляционных базах данных**, где это означает, что **таблицы разбиваются по столбцам**, а затем **распределяются по нескольким машинам**

Горизонтальная распределенность

* В этом случае клиент или сервер могут одновременно исполняться на одном и том же узле, разделяя между собой его ресурсы.
* Системы поддерживающие горизонтальную распределенность получили название пиринговых систем (**peer-to-peer systems**)
* Все процессы, составляющие одноранговую систему, равны. Это означает, что функции, которые необходимо выполнить, представлены на каждом узле отдельным процессом, составляющим распределенную систему.
* Как следствие, большая часть взаимодействия между процессами является симметричной:
  + каждый узел будет одновременно действовать и как клиент и как сервер

Задачи решаемые с помощью р2р систем (1)

1. ***Уменьшение/распределение затрат****.* Серверы централизованных систем, которые обслуживают большое количество клиентов, обычно несут на себе основной объем затрат ресурсов (денежных, вычислительных и др.) на поддержание вычислительной системы. P2P архитектура может помочь распределить эти затраты между узлами сети. Так как узлы, как правило, автономны, важно, чтобы затраты были распределены справедливо.
2. ***Объединение ресурсов.*** Каждый узел в P2P-системе обладает определен- ными ресурсами (вычислительные мощности, объем памяти). Приложения, которым необходимо большое количество ресурсов, например ресурсоза- тратные задачи моделирования или распределенные файловые системы, используют возможность объединения ресурсов всей сети для решения своей задачи. При этом важны как объем дискового пространства для хранения данных, так и пропускная способность сети.
3. ***Повышенная масштабируемость****.* Поскольку в сетях Р2Р отсутствует сильный центральный механизм, важной задачей является повышение масштабируемости и надежности системы. Масштабируемость определяет количество систем, которые могут быть достигнуты из одного узла, сколько систем могут функционировать одновременно, сколько пользователей может пользоваться сетью, сколько памяти может быть использовано.
4. ***Надежность сети*** определяется такими параметрами как количество сбоев в работе сети, отношение времени простоя к общему времени работы, доступностью ресурсов и т.д. Таким образом, основной проблемой становится разработка новых алгоритмов обнаружения ресурсов, на которых базируются новые P2P платформы
5. ***Анонимность.***Бывает, пользователь не желает, чтобы другие пользователи или поставщики услуг знали о его нахождении в сети. При использовании **центрального сервера трудно обеспечить анонимность**, так как серверу, как правило, необходимо идентифицировать клиента, по крайней мере через интернет адрес. При использования P2P-сети пользователи могут избежать предоставления любой информацию о себе

Принципы построения РС Р2Р

* В распределенной системе Р2Р **каждый узел** участник **знает** некоторое число логических **соседей** с которыми он может поддерживать обмен напрямую, без посредников, посылая и получая в ответ сообщения по сети.
* Этот набор соседей формирует логический граф связности для всех узлов системы. Это граф часто называют **наложенной** сетью Р2Р (overlay network).
* Системы Р2Р должны соответствовать следующим критериям:
* **Самооганизация:** система самостоятельно адаптируется к подключению или отключению узла к системе. Пиры используют локальную информацию получаемую от своих соседей для организации взаимодействия друг с другом.
* **Распределенность:** отсутствует централизованное управление поведением узла в системе.
* **Масштабируемость:** система может масштабироваться неограниченно избегая таким образом проблем «бутылочного горла», отказов отдельных узлов и проблем перегрузки узлов.

1. Классификация P2P по степени централизации.

Классификация Р2Р систем по степени их Централизации

* Централизованные P2P системы
* Иерархические
* Гибридные
  + Системы пограничных серверов
  + Системы основанные на сотрудничестве пиров
* Децентрализованные P2P системы

Централизованные Р2Р системы

* В этом случае некоторая **группа узлов** отвечает за выполнение критически **важных для всей системы** операций, например:
  + Аутентификация пиров в системе.
  + Определение расположения узлов и ресурсов принадлежащих узлам.
* Эти управляющие узлы могут располагаться в одной LAN, но могут быть разбросаны географически, но в любом случае между ними должны существовать широкополосные каналы связи.

Пример: **Napster**

* Узел S посылает запрос на определение места размещения некоторых данных к

управляющим узлам (core nodes).

* На основе имеющегося у них глобального индекса управляющий узел форвардирует запрос к узлам А и В.
* Узлы А и В отвечают напрямую S.
* При отправке запроса S устанавливает таймер, по истечение которого, он выбирает узел из числа ответивших ему.

Недостатки централизованных P2P систем

* Управляющие сервера являются потенциальными точками отказа.
* Такое решение является плохо масштабируемым.
* Эти недостатки отсутствуют в частично централизованных (гибридных) системах (рассматриваются в разделе гибридные архитектуры Р2Р систем).

Иерархически организованные Р2Р системы

* В **не структурированных системах** по мере их роста поиск данных может стать невозможен. Источник этой проблемы масштабируемости очевиден – **отсутствие** детерминированой **процедуры маршрутизации** запросов **поиска ресурсов**.
* **В качестве альтернативы** во многих неструктурированных системах **создаются специальные узлы хранящие индекс** экземпляров данных имеющихся в системе.
* Имеются и другие ситуации в которых отказ от симметричной природы р2р систем имеет смысл. Например, сети доставки контента (CDN – Content Delivery Network).
* В CDN сетях узлы системы хранят копии Web документов отдельных сайтов Интернет к которым пользователям требуется быстрый доступ. Для поиска узлов используются **брокеры**, которые хранят сведения о степени используемости ресурсов и **доступности узлов**, что позволяет выбирать наиболее подходящий из узлов для получения ресурсов.

Иерархическая организация сети Р2Р

* Узлы играющие роль брокеров, а также узлы поддерживающие индекс ресурсов называются **супер-узлами.** Супер-узлы часто образуют сеть р2р. Обычные узлы выполняющие роль клиентов супер-узлов называют **слабыми** (**weak nodes**).
* В этих сетях **супер-узлы должны всегда** оставаться **доступными** в сети. Это создает проблемы надежности, которые можно компенсировать, путем развертывания системы копирования восстановления между парами супер-узлов, и обеспечением подключения рядовых узлов к обеим супер-серверам.
* В сетях с супер-узлами, возникают и другие проблемы, например **как отбирать узлы** на роль **супер-узлов**.

Иерархическая р2р система.   
Пример: SKYPE

1. Гибридные архитектуры Р2Р систем. Системы с граничными серверами. Архитектура bittorrent.

Гибридные архитектуры р2р систем

* Существует множество архитектур РС в которых успешно сочетаются архитектуры клиент сервер и децентрализованные архитектуры Р2Р систем.

Частично централизованные (гибридные) системы Р2Р

* В этих системах все узлы делятся на несколько типов в зависимости от иерархической организации. В двухуровневой системе имеется два вида узлов:
  + суперузлы (supernodes); мощные узлы связанные между собой хорошими каналами связи. Хранят метаданные о расположении ресурсов и
  + рядовые пиры (peers); связаны с одним или несколькими суперузлами.

Пример Kazaa:

1. Узел S направляет к ближайшему суперузлу (SP) запрос на поиск ресурса. Суперузел форвардирует это запрос ко всем подключенным узлам.
2. Узел А отвечает на запрос напрямую к S.
3. Так как запрос форвардируется и к другим суперузлам, то него отвечает и узел С.
4. S выбирает сам к какому из ответивших узлов обраться для получения ресурса.

Системы с граничными серверами

* Эти системы разворачиваются в Интернет, при этом на границах сетей сервис провайдеров (ISP – Internet Service Provider) размещаются сервера этих систем.
* Конечные пользователи этих систем располагаются в сетях сервис провайдеров и подключаются к распределенной системе через граничные сервера.
* Основное назначение **граничных серверов** предоставлять контент для пользователей, возможно после предварительной фильтрации и перекодировки.
* Базовая модель состоит в том, что для организации **граничный сервер является источником всего контента**, получаемого из Интернет. Этот сервер может использовать другие граничные сервера, например для репликации Web страниц. страниц.
* Cистемы c граничными серверами назодят широкое применение в системах Интернет вещей (IoT), и облачных системах сенсоров.

Архитектура BitTorrent

* 1. Если узел хочет опубликовать файл или набор файлов, то программа-клиент BitTorrent сети разделяет передаваемые файлы на части и создает ***файл метаданных*** (идентификатор раздачи), который содержит следующую информацию:
  + URL **трекера** (супер-сервера ресурса, который хранит актуальную информацию об активных узлах которые имеют отдельные части файла);
  + **Общая информация о файлах** (имя, длина и пр.);
  + **Хеш-суммы** SHA1 сегментов раздаваемых файлов;
  + **Passkey** пользователя – **ключ**, который однозначно определяет пользователя загрузившего файл;
  + **Хеш-суммы файлов** целиком (не обязательно);
  + Альтернативные источники – адреса **альтернативных трекеров**, на которых можно найти информацию по данному файлу (не обязательно).

**BitTorrent протокол**

* Сам протокол состоит в следующем:
* есть небольшой торрент-файл, содержащий информацию о **трекере**, который следит за файлом, и список сегментов, из которых состоит файл:
* для сегмента хранится 160-битовый SHA-1 соответствующего сегмента файла (обычный размер сегмента -- 64-512 кб).
* Таким образом, скачав файл, можно посчитать от него хеши и проверить, все ли скачалось без ошибок.
* Кроме торрент-файла есть **трекер**: сервер, который следит за состоянием пиров -- узлов, имеющих отношение к раздаче.
* Пир, получив торрент-файл, подключается к трекеру и узнает список **сидов** -- **пиров**, у которых уже есть некоторые сегменты файла, **готовые их отдать**.
* Узнав **адрес сида**, у которого можно что-то скачать, клиент уже **обращается напрямую к нему и скачивает у него файл**. Таким образом, все **данные идут напрямую от пира к пиру** (peer-to-peer), а **трекер** участвует лишь в качестве **координатора процесса**.
* Протокол BitTorrent был разработан в 2001 году **Коэном Брахмом**, чтобы позволить набору узлов быстро и легко обеспечивать общий доступ к файлам.

Алгоритм загрузки документа Bittorrent

* Алгоритм загрузки документа производится следующим образом:

2. клиент подключается к трекеру по URL из файла метаданных и

* + сообщает хеш-идентификатор требуемого файла;
  + получает адреса пиров скачивающих и раздающих данный файл;

3. клиенты соединяются между собой и **обмениваются информацией без участия трекера**.

* В последнее время стала распространяться **альтернативная технология поиска** и загрузки документов на основе «**магнитных ссылок**» (magnet links) и подхода распределенных хеш-таблиц (Distributed Hash Table — **DHT**) по сути дела представляющих собой **реализацию алгоритма маршрутизации** документов.
* Причина возникновения этой технологии – дальнейшее развитие деперсонализации и **попытка торрент-трекеров защититься** от юридического **преследования правообладателей**. Торрент-файл для такой раздачи **создаётся без адреса трекера** и клиенты находят друг друга через распределенные хеш-таблицы.

1. P2P системы на основе технологии распределенного реестра. Понятие распределенного реестра (DLT). Виды распределенных реестров. Классификация распределенных баз данных. Пример использования DLT.

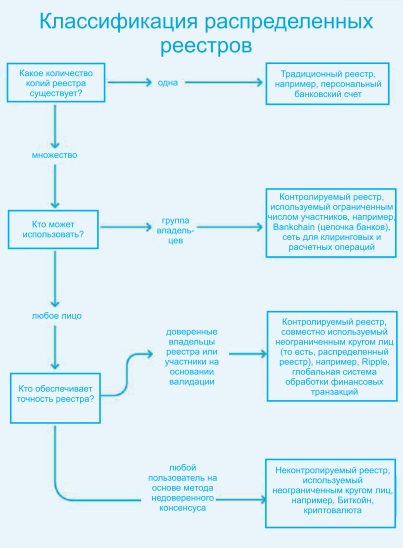
Понятие распределенного реестра

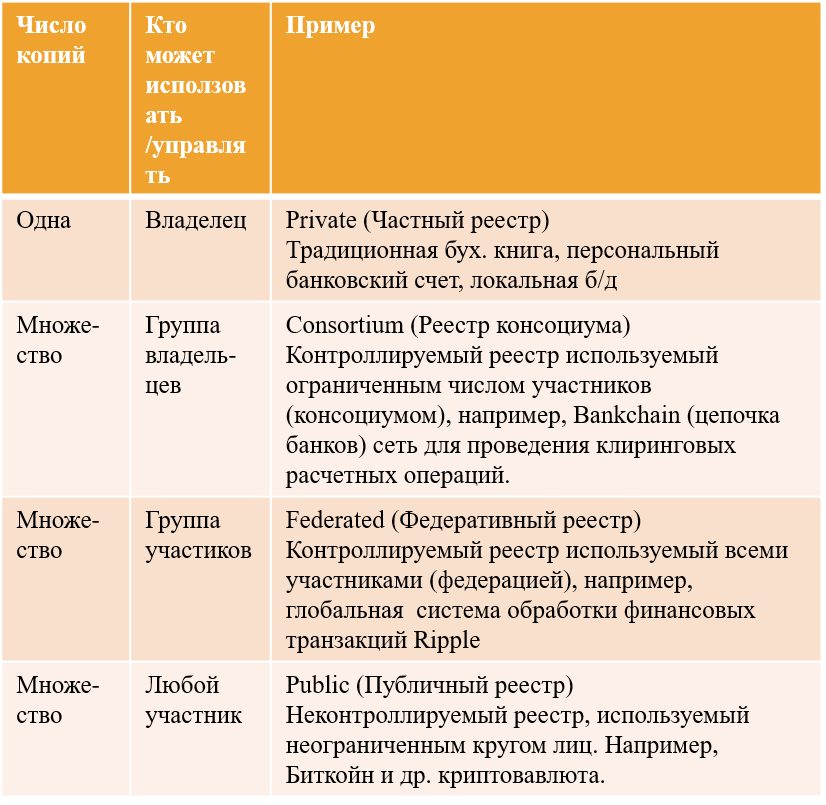
* Ledger – в переводе с англ. означает, бухгалтеская книга, главная книга, гросбух, регистрационный журнал, **реестр**.
* Распределенный реестр (DL – Distributed Ledger) или распределенная база данных записи которой хранятся одновременно на всех узлах сети Р2Р.
* Распределенный реестр (DL) имеет несколько ключевых особенностей:
  + отсутствие **центрального администратора** и **центральной** (генеральной) **копии** б/д;
  + **совместное использование данных всеми узлами** с синхронизацией данных по заданному алгоритму;
  + **децентрализованное географическое распределение копий базы** данных между всеми узлами р2р сети.
* Каждый узел вносит изменения в реестр независимо от других узлов, затем все они голосуют за внесение изменений и при достижении консенсуса реестр дополняется новыми данными. Каждый участник сети при этом обладает собственной идентичной копией реестра, а сами изменения добавляются в течение нескольких минут.
* В отличие от традиционных баз данных, распределительные реестры намного лучше защищены от атак и несанкционированного изменения данных, поскольку копии географически отдалены друг от друга и для хакерского изменения потребуется произвести атаку сразу на все узлы связи. Кроме того, ноды обнаружат несанкционированные изменения и не поддержат внесение данных.

Виды распределенных реестров

* Условно все **виды распределенных реестров** можно разделить на:
* **Публичные.** Применяются в большинстве криптовалют и представляют собой базу данных с открытым исходным кодом. Они работают на алгоритмах подтверждения выполнение некоторого объема вычислений. В такой системе каждый участник может загрузить себе на локальное устройство базу данных и участвовать в согласованном процессе внесения изменений. Также любой желающий может просмотреть всю добавленную информацию.
* **Федеративные.** Федеративные базы данных работают под управлением группы людей. В отличие от открытых реестров они не поддерживают внесение новых данных всеми желающими. Процесс изменения реестра контролируется исключительно заранее выбранными узлами сети. Применяются они преимущественно в банковском секторе и обеспечивают большую конфиденциальность.
* **Частные.** Право на внесение изменений в такой реестр имеет только определенная централизованная организация. Информация может быть открыта для публичного чтения или быть ограниченной в произвольной степени. Частные распределительные реестры, как правило, используются компаниями для хранения внутренней информации и проведения аудита. Такие системы более уязвимы, нежели публичный блокчейн, но позволяет модернизировать устаревшие системы хранения информации в компаниях.

Классификация распределенных баз данных





Где используются DLT ?

* Основной областью использования распределительных реестров сейчас остаются **криптовалюты**, где по такой системе хранится информация о транзакциях и производится подсчет баланса кошельков пользователей.
* Полностью понимая технологию правительство и частный сектор смогут существенно модернизировать процесс хранения и использования информации. Например, **правительства** многих стран постепенно **внедряют технологию DLT** в разные секторы своей работы.
* Например, в Эстонии несколько лет ведется разработка **KSI (Keyless signature infrastructure)** на основе распределительного реестра, которая позволит гражданам проверять достоверность сведений в государственных базах данных. Цель проекта — **защита** свыше 1 млн **медицинских записей**.
* Каждый раз, когда данные подвергаются любому изменению, система автоматически создает обновленную подпись, позволяя оперативно отреагировать в случае неправомерного посягательства на целостность информации. Сами записи не хранятся в реестре KSI — фиксируются лишь серии их хеш-значений, в которых и отображается информация об изменениях. **Инфраструктура цепочек блоков KSI** была реализована еще в апреле 2008 года **до появления** технологии **блокчейна** Биткойна.
* Правительство Англии создало комитет по исследованию технологии DLT и разработке государственных проектов на его основе.

1. Классификация Р2Р систем по их Структуре. Структурированные Р2Р системы. Индекс ресурса структурированной P2P системы. Распределенная хэш таблица (DHT). Поиск узла в структурированной P2P системе. Поиск с помощью маршрутизации документов.

Структурированные Р2Р системы

* Примерами систем, использующих структурированные оверлейные сети являются:
  + Chord [23];
  + CAN [34];
  + Pastry [35];
  + Tapestry [35];
  + Kademlia [37];
  + Viceroy [38];
  + P-Grid [39];
  + SkipNet [40].
* Структура Р2Р системы определяется структурой наложенной сети (overelay network), которая может иметь одну из известных топологий:
  + кольцо; двоичное дерево; решетка; и т.п.
* Топология наложенной сети используется для эффективного поиска данных (ресурсов) расположенных на узлах системы.
* Для структурированных одноранговых систем характерно то, что они обычно основаны на использовании так называемого **бессемантического** индекса.
* Это означает, что каждый **элемент данных**, который должен поддерживаться системой, однозначно **связан с ключом**, и что этот **ключ** впоследствии используется в качестве **индекса**, для **поиска** данных.

Индекс ресурса структурированной P2P системы

* Важной характеристикой структурированной Р2Р системы является так называемый **индекс ресурса**, который однозначно определяет **каждый экземпляр данных** поддерживаемый в системе и соответственно **каждый узел** располагающий каким-либо ресурсом.
* **Соседство в системе** определяется по близости значений индексов у узлов.
* В качестве такого индекса используется ключ хэш функции:

***Key(data item) = hash(data item's Value).***

Распределенная хэш таблица (Distributed Hash Table)

* Каждому узлу структурированной P2P системы назначается идентификатор из одного того же набора значений хэша. И каждый узел становится ответственным за хранение данных, связанных (ассоциированных) с определенным набором ключей.
* По сути, система описывается с помощью таблицы хэшей - распределений хэш таблицы (DHT - Distributed Hash Table).

Поиск узла структурированной P2P системы

* Для того чтобы найти узел, ассоциируемый с ключом, необходимо выполнить операцию поиска узла по ключу:

***node = lookup (key).***

* В задаче поиска топология наложенной сети играет ключевую роль.
* Любой узел однозначно связан с некоторым ключом, что позволяет организовать эффективную маршрутизацию для поиска узла, ответственного за хранение ресурса.

ПОИСк с помощью маршрутизации документов (1)

* Данный метод обеспечивает поиск документов без участия центрального индекса, средствами самой вычислительной сети.
* В основе данного метода лежит **принцип присвоения уникальных идентификаторов** каждому **узлу** вычислительной сети, а также каждому **ресурсу** (документу, сервису и др.), который данная сеть может предоставлять.

Изображение выглядит как диаграмма, круг, зарисовка, План

Автоматически созданное описание

Когда какой-либо пир (на рисунке – узел 7008) производит публика- цию ресурса в сети, каким-либо образом вычисляется идентификатор данного ресурса (на рисунке – документ 0110). При этом, идентификаторы узлов сети и ресурсов имеют единую область возможных значений.

Далее, копия данного ресурса (или ссылка на него) **связывается** с узлом, который имеет наиболее похожий идентификатор, на рисунке показана трасса документа 0110:

узлы 7008 – 0459 – 0009 – 0040.

Данная процедура производится следующим образом:

1. Производится сравнение идентификатора ресурса с идентификаторами всех соседних узлов.
2. Если идентификатор *текущего* узла ближе всего (по некоторой метрике) к идентификатору документа, то процесс трассировки завершается.

Если один из идентификаторов соседних узлов ближе к идентификатору документа, чем идентификатор текущего узла, то ссылка на данный ресурс *копируется* на узел с более близким идентификатором и процесс повторяется с п. 1. (узел 0040 на рисунке).

* Поиск ресурса производится по аналогичному алгоритму, но вместо копирования документа происходит трансляция запроса (в запросе содержится идентификатор запрашиваемого ресурса, например 0110 на рисунке 57) от узла, инициировавшего запрос (на рисунке – узел 5203) к узлу, идентификатор которого ближе всего соответствует идентификатору запрашиваемого документа.
* Соответственно, в процессе трансляции данного запроса должен появиться такой узел, который хранит информацию об интересуемом документе, если данный документ находится в соответствующей части сети.
* К недостаткам такого подхода можно отнести необходимость знания точного идентификатора документа, который необходимо найти в сети (невозможность поиска по отдельным атрибутам документа), а также возможность образования «островов», затрудняющих алгоритм поиска.

1. Неструктурированные P2P системы. Методы поиска узлов в неструктурированных Р2Р системах. Метод затопления. Метод случайного блуждания. Метод поиска, основанный на политике.

Неструктурированные Р2Р системы

Примерами систем, использующих неструктурированные оверлейные сети являются:

* Freenet;
* Gnutella;
* FastTrack;
* BitTorrent;
* UMM;
* Gia;
* Phenix.
* В неструктурированных сетях P2P, таких как Gnutella, не существует никаких правил, которые определяют местоположение хранящихся данных, а **топология сети произвольна**.
* Каждый узел в такой системе поддерживает **список соседних узлов**, а об остальных он ничего не знает.
* В результате наложенная сеть такой системы представляет собой случайный граф: - граф в котором дуга между вершинами <u,v> существует с некоторой вероятностью **P[<u,v>]**. В идеале эта вероятность имеет одно и тоже значение, для всех пар вершин.
* При присоединении узла к Р2Р системе, он контактирует с **общеизвестными в системе узлами** и получает от них список других пиров (**соседей**) от которых он может узнать о еще большем числе пиров.
* В таких системах, если пир хочет найти какой-либо ресурс, то он не может следовать какой-то определенной процедуре маршрутизации. Он должен прибегнуть к специальной **процедуре поиска** установленной в этой системе.
* Имеется два предельных метода поиска в таких системах:
  + Метод затопления (flooding).
  + Метод случайного блуждания (random walk).

Метод затопления  
(лавинная рассылка)

* Узел **u** посылает запросы ко всем известным ему соседям.
* Если узел **v** имеет требуемые данные, то он либо посылает их напрямую узлу породившему запрос, либо отсылает их к узлу от которого он получил запрос.
* Если узел не имеет данных, то он форвардирует запрос ко всем своим соседям.
* Если узел уже получал такой запрос, то он не будет повторно на него отвечать.
* Для того, чтобы избежать сильного затопления системы, для запросов устанавливается время жизни (TTL-Time to live). Каждый промежуточный узел обрабатывающий запрос уменьшает TTL на 1. Когда TTL станет равным 0 запрос удаляется из системы.
* От выбора значения TTL сильно зависит эффективность поиска и степень затопления системы запросами (см. формулу).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Метод случайного блуждания

* Узел **u** просто опрашивает соседей выбирая их **случайным образом**. Например, пусть u выберет для опроса узел **v.**
* Если **v** имеет требуемые данные, то он напрямую ответит u, если нет, то v форвардирует запрос своему соседу, выбранному случайным образом.
* Очевидно, что этот метод порождает значительно **меньший трафик**, но для поиска нужного ресурса может потребоваться **значительно больше** времени.
* Для уменьшения времени ожидания узел источник процедуры поиска может одновременно стартовать **n** процедур случайного блуждания. В этом случае в этом случае время поиска снижается в n раз.
* Случайное блуждание также требует своей принудительной остановки. Для этого можно также использовать TTL, либо можно установить условия при которых узел принявший запрос сам определит нужно форвардировать запрос случайным образом далее или нет.
* Для неструктурированных систем также необходимо определить порядок сравнения для определения признака нахождения искомых данных. Это могут быть те же методы, которые применяются в структурированных системах Р2Р.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Метод поиска основанный на политике

* Этот метод занимает промежуточное положение между методами затопления и случайного блуждания.
* В этом методе для выбора узлов в качестве соседей используется некоторая установленная политика.
* Например:
  + в качестве узлов к которым перенаправляются запросы выбираются узлы, которые **ранее более успешно выполняли поиск**, по сравнению с другими узлами.
  + в качестве узлов к которым направляются запросу выбираются узлы, имеющие **наибольшее число соседей**.
  + и т.п.

1. Понятие процесса выполнения. Составляющие контента процесса. Жизненный цикл процесса в ОС. Системные вызовы управляющие процессами. Смена режимов работы процессора при выполнении системного вызова. Виды системных вызовов, связанных с процессами. Идентификация процессов в системе. Дерево процессов.

**Процессы выполнения**

* Для выполнения программ операционная система создает несколько виртуальных процессоров, по одному для каждой программы.
* Чтобы отслеживать эти виртуальные процессоры, операционная система поддерживает ***таблицу процессов*** *(process table), содержащую записи для сохранения значений регистров* процессора, карт памяти, открытых файлов, учетных записях пользователей, привилегиях и т. п.
* *Процесс (process) часто определяют как выполняемую программу,* то есть **программу**, которая в настоящее время выполняется на одном из виртуальных процессоров операционной системы.

**Понятие процесса**

* **Процессом**, называют программу в момент выполнения, в некоторых ОС исполняемую программу называют **задачей**, процесс и задача являются синонимами.
* С каждым процессом связывается его **адресное пространство** — список адресов в памяти от некоторого минимума (обычно нуля) до некоторого максимума, которые процесс может прочесть и в которые он может писать. Адресное пространство содержит саму программу, данные к ней и ее стек.
* Со всяким процессом связывается некий **набор регистров**, включая счетчик команд, указатель стека и другие аппаратные регистры, плюс вся остальная информация, необходимая для запуска программы. Это **информация о процессе**.
* Вся совокупность информации о процессе, необходимая для ее продолжения процессором после прерывания называется **контекстом** процесса. Контекст процесса является частью (подмножеством) информации о процессе.
* Всякий процесс выполняемый в системе запускается от **имени пользователя** инициировавшего запуск программы на выполнение.
* **Права доступа процесса** к ресурсам системы определяются **правами доступа пользователя**, от чьего имени программа была запущена.

**Составляющие контента процесса**

* В режиме выполнения:
  + адресное пространство процесса в оперативной памяти ЭВМ;
  + информация о процессе и ресурсах которые он использует, например об открытых файлах или установленных сетевых соединениях.
* В режиме останова:
  + образ памяти процесса (адресное пространство процесса);
  + информация о процессе хранящаяся в таблице процессов.

**Жизненный цикл процесса в ОС**

* Включает в себя следующие стадии:
  + Создание процесса;
  + выполнение процесса;
  + уничтожение процесса.

**Системные вызовы управляющие процессами**

* Процесс создается родительским процессом с помощью обращения к функции ядра «**создать процесс**».
* Обращение к функциям ОС называется системным вызовом.
* Главными системными вызовами, управляющими процессами, являются вызовы связанные с созданием и уничтожением процессов.
* Системный вызов выполняется в привилегированном режиме работы процессора.

**Смена режимов работы процессора при выполнении системного вызова**



**Виды системных вызовов связанных с процессами**

* создание процесса;
* освобождение или выделение дополнительной памяти процессу;
* ожидание завершения какого-либо процесса;
* перекрытие адресных пространств процесса;
* передача сигнала процессу;
* завершение процесса;
* и др.

**Идентификация процессов в системе**

* Для идентификации процессов в системе используются идентификаторы процессов PIDs (Process Identifier)
* **PID** – это число присваемое процессу при запуске.
* Каждому процессу присваевается идентификатор пользователя (UID – User Idetifier) и GID (Group Identifier) запустившего данный процесс.

Дерево процессов в ОС

* Если процесс способен создавать несколько других процессов (называющихся **дочерними процессами**), а эти процессы в свою очередь могут создавать собственные дочерние процессы, то в системе образуется дерево процессов.
* В системе имеется процесс с PID=1, который является прародителем всех процессов.

Изображение выглядит как зарисовка, круг, диаграмма, рисунок

Автоматически созданное описание

1. Связанные процессы и межпроцессное взаимодействие. Взаимоблокировка процессов. Методы межпроцессного взаимодействия. Сигналы, передаваемые процессам. Семафоры и мьютексы. Недостатки процессов.

**Связанные процессы и межпроцессное взаимодействие**

* Связанные процессы — это те, которые объединены для выполнения некоторой задачи, и им нужно часто передавать данные от одного к другому и синхронизировать свою деятельность.
* Такая связь называется межпроцессным взаимодействием.
* **Межпроцессное взаимодействие** (Inter-process communication (IPC)) — это набор методов для обмена данными между потоками процессов. Процессы могут быть запущены как на одном и том же компьютере, так и на разных, соединенных сетью.

**Взаимоблокировка процессов**

* Когда взаимодействуют два или более процессов, они могут попадать в патовые ситуации, из которых невозможно выйти без посторонней помощи.
* Такая ситуация называется тупиком, тупиковой ситуацией или взаимоблокировкой.

Методы межпроцессного взаимодействия (1)

* Существуют следующие методы межпроцессного взаимодействия:
  + **Каналы** (pipe) средство связи стандартного вывода одного процесса со стандартным вводом другого. Каналы старейший из инструментов IPC, существующий приблизительно со времени появления самых ранних версий оперативной системы UNIX.
  + **Сигналы**. Сигналы являются программными прерываниями, которые посылаются процессу, когда случается некоторое событие. Сигналы могут возникать синхронно с ошибкой в приложении, например SIGFPE (ошибка вычислений с плавающей запятой) и SIGSEGV (ошибка адресации), но большинство сигналов является асинхронными.
  + **Очереди сообщений.** Очереди сообщений представляют собой связный список в адресном пространстве ядра. Очереди сообщений как средство межпроцессной связи позволяют процессам взаимодействовать, обмениваясь данными. Данные передаются между процессами дискретными порциями, называемымисообщениями. Процессы, использующие этот тип межпроцессной связи, могут выполнять две операции:послать или принять сообщение.
  + **Семафоры и мьютексы.** Семафор (semaphore) - это целая переменная, значение которой можно опрашивать и менять только при помощи неделимых (атомарных) операций. Двоичный семафор может принимать только значения 0 или 1. Вычислительный семафор может принимать целые неотрицательные значения. **Мьютекс** - отличается от [семафора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) тем, что только владеющий им [поток](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) может его освободить.
  + **Разделяемая память.** Разделяемая память может быть наилучшим образом описана как отображение участка (сегмента) памяти, которая будет разделена между более чем одним процессом. Это гораздо более быстрая форма IPC, потому что здесь нет никакого посредничества (т.е. каналов, очередей сообщений и т.п.).
  + **Сокеты.** Сокеты обеспечивают двухстороннюю связь типа точка-точка *между двумя процессами.* Они являются основными компонентами межсистемной и межпроцессной связи. Каждый сокет представляет собой конечную точку связи, с которой может быть совмещено некоторое имя. Он имеет определенный тип, и один процесс или несколько, связанных с ним процессов.

**Сигналы передаваемые процессам**

* Сигналы являются программными аналогами аппаратных **прерываний** и могут быть сгенерированы по различным причинам, а не только из-за истечения какого-либо интервала времени.
* Многие **аппаратные прерывания** (например, вызванные выполнением недопустимой команды или использованием неправильного адреса) также **преобразуются в сигналы процессу**, в котором произошла ошибка.
* Сигнал вызывает:
  + временную **остановку работы** процесса независимо от того, что процесс делает в данный момент;
  + **сохраняет** его **регистры** в стеке
  + запускает специальную **процедуру обработки сигнала** (например, передающую повторно предположительно потерянное сообщение).
* **После завершения обработки** сигнала работающий процесс запускается **заново в том состоянии**, в котором он находился до сигнала.

**Семафор**

* **Семафо́р** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *semaphore*) — примитив синхронизации[[1]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) работы [процессов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [потоков](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F), в **основе которого лежит счётчик**, над которым можно производить две [атомарные операции](https://ru.wikipedia.org/wiki/%D0%90%D1%82%D0%BE%D0%BC%D0%B0%D1%80%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F): увеличение и уменьшение значения на единицу, при этом операция уменьшения для **нулевого значения** счётчика является **блокирующейся**[[2]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Служит для построения более сложных механизмов синхронизации[[1]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и используется **для синхронизации параллельно работающих задач,** для **защиты передачи данных** через [разделяемую память](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D0%BC%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), для **защиты** [критических секций](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D0%B5%D0%BA%D1%86%D0%B8%D1%8F), а также **для управления доступом к аппаратному обеспечению**.
* **Легковесный семафор** — механизм, позволяющий в ряде случаев **уменьшить количество системных вызовов** за счёт использования активного цикла ожидания в течение некоторого времени перед выполнением блокировки[[3]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).
* Семафоры могут быть **двоичными** и **вычислительными**.
  + **Вычислительные семафоры** могут принимать **целочисленные неотрицательные** значения и используются для работы с ресурсами, количество которых ограничено, либо участвуют в синхронизации параллельно исполняемых задач.
  + **Двоичные семафоры** являются частным случаем вычислительного семафора и могут **принимать** только два **значения: 0 и 1**

**Мьютекс**

* **Мью́текс** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *mutex*, от *mutual exclusion* — «взаимное исключение») — аналог одноместного [семафора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), служащий в [программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) для [синхронизации](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) одновременно выполняющихся [потоков](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F).
* Мьютекс отличается от [семафора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) тем, что **только владеющий** им [поток](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) может **его освободить**, т.е. перевести в отмеченное состояние. Мьютексы — это один **из вариантов семафорных** механизмов для организации взаимного исключения. Они реализованы во многих [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), их **основное назначение** — организация **взаимного исключения для потоков** из одного и того же или из разных процессов.
* Цель использования мьютексов — **защита данных** от повреждения в результате **асинхронных изменений** ([состояние гонки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8)), однако могут порождаться другие проблемы — например [взаимная блокировка](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) (клинч).

**Недостатки процессов**

* Хотя процессы являются строительными блоками распределенных систем, практика показывает, что дробления на **процессы**, предоставляемого операционными системами, на базе которых строятся распределенные системы, **недостаточно**.
* Вместо этого оказывается, что наличие более тонкого дробления в форме нескольких ***потоков выполнения (threads) на процесс значительно упрощает построение*** распределенных приложений и позволяет добиться лучшей производительности.

1. Понятие потока выполнения. Преимущества потоков выполнения. Реализация потоков на пользовательском уровне и в ядре. Преимущества потоков выполнения на пользовательском уровне. Преимущества использования потоков против группы конкурирующих процессов.

Понятие потока выполнения

* **Основная причина** использования потоков заключается в том, что во многих приложениях **одновременно происходит несколько действий**, часть которых может периодически **быть заблокированной**.
* Модель программирования упрощается за счет разделения такого приложения на несколько **последовательных потоков**, выполняемых в **квазипараллельном** режиме.

Преимущества потоков выполнения

1. **Возможность использования** параллельными процессами **единого адресного пространства и всех имеющихся данных**. Эта возможность играет весьма важную роль для тех приложений, которым не подходит использование нескольких процессов (с их раздельными адресными пространствами).
2. Легкость (то есть быстрота) их создания и ликвидации по сравнению с более «тяжеловесными» процессами. Во многих системах **создание потоков осуществляется в 10–100 раз быстрее**, чем создание процессов.
3. Когда потоки работают в **рамках одного центрального процессора**, они не приносят никакого прироста производительности, но когда выполняются значительные вычисления, а также **значительная часть времени тратится** на ожидание ввода-вывода, наличие потоков позволяет этим действиям **перекрываться** во времени, ускоряя работу приложения.
4. И наконец, потоки весьма полезны для систем, имеющих **несколько** центральных процессоров, где есть реальная возможность параллельных вычислений.

**Реализация пакетов потоков выполнения**

* Потоки выполнения обычно существуют в виде **пакетов потоков**.
* Подобные **пакеты содержат механизмы** для создания и уничтожения потоков, а также для работы с переменными синхронизации, такими как мьютексы и условные переменные.
* Существует два основных подхода к реализации пакетов для потоков выполнения.
  + Первый из них состоит **в создании библиотеки для работы с потоками** выполнения, выполняющейся исключительно в режиме пользователя.
  + Второй подход предполагает, что **за потоки выполнения отвечает** (и управляет ими) **ядро**.
* В модели Many-to-one threading - несколько потоков отображаются на один планируемый процесс.
* В результате при использовании системного вызова блокирующего данный процесс **блокируются все потоки** связанные с данным процессом.

Преимущества потоков выполнения на пользовательском уровне (1)

* Библиотека для работы с потоками выполнения на пользовательском уровне имеет множество преимуществ.
* **Во-первых,** **дешевле** обходится создание и уничтожение потоков выполнения.
  + Поскольку все управление потоками реализуется в адресном пространстве пользователя, стоимость создания потока выполнения определяется в первую очередь **затратами на память**, выделяемую для создания стека под поток.
  + Аналогично и **уничтожение потока** выполнения в основном состоит в освобождении памяти, задействованной под стек, после того как надобность в потоке отпадает. Обе операции достаточно дешевы.
* **Второе преимущество** потоков выполнения на пользовательском уровне состоит в том, что **переключение контекста** требует всего **нескольких** инструкций.
  + В основном **в сохранении** и последующем восстановлении сохраненных значений при переключении с потока на поток **нуждаются** **исключительно** в операциях со значениями **регистров процессора**.
  + Нет необходимости **изменять** **карты памяти**, сбрасывать **буфер TLB**, контролировать **загрузку** процессора и т. д.
  + **Переключение контекста** потоков выполнения производится при необходимости в **синхронизации двух потоков**, например, при обращении к секции совместно используемых данных.

Реализация потоков в ядре

* Для потоков реализуемых на уровне ядра вся информация о потоках аналогична той, что используется для пользовательских потоков.
* При работе в режиме ядра создание и уничтожение потоков требует **более существенных затрат.**
  + Хотя потоки на уровне ядра могут решить многие проблемы, но их **главный недостаток** в весьма существенных **затратах на реализацию системных вызовов**, поэтому если потоки создаются/удаляются достаточно часто, то это влечет за собой существенные издержки.
  + Другой проблемой являются **сигналы**. Сигналы предназначены для процессов, а не для потоков.
* Однако в свете того факта, что **производительность переключения контекста** обычно определяется **неэффективным использованием кешей** памяти, а не различием между потоковой моделью «многие к одному» или «один к одному», многие операционные системы сейчас предлагают последнюю модель хотя бы из-за ее простоты.

**Использование потоков против применения группы конкурирующих процессов**

* Применение потоков является способом **одновременного** и **параллельного** исполнения в рамках **одного приложения**.
* На практике, часто можно встреть реализации когда приложения строятся как **коллекция параллельно работающих процессов**, объединяемых с помощью средств межпроцессного взаимодействия предлагаемых операционными системами.
  + Примером применения такого подхода является реализация веб-сервера Apache, который по-умолчанию стартует как пятерка процессов, предназначенных для обслуживания поступающих запросов.
  + **Каждый процесс является *однопоточной* реализацией** сервера, способной взаимодействовать с другими экземплярами с помощью стандартных средств ОС.
* Использование процессов вместо потоков имеет **одно важное преимущество** – обеспечивается **разделение пространств** данных процессов.

Разделение процессов и их пространств данных обеспечивается аппаратными средствами процессоров.

* Это преимущество нельзя недооценивать, так как при использовании потоков вся забота о параллельном доступе к разделяемым данным **ложиться на разработчика** многопоточного приложения.

1. Понятие виртуализации. Виды интерфейсов в компьютерной системе. Принципы виртуализации. Способы виртуализации. Виртуализация ЭВМ. Виртуализация ресурсов физического сервера. Область эффективной виртуализации. Роль виртуализации в распределенных системах.

Понятие виртуализации

* Для описания работы вычислительной системы необходимы **три класса** фундаментальных абстракций:
  + интерпретаторы, память и каналы связи.
* Им соответствуют следующие физические реализации:
  + процессоры для преобразования информации;
  + первичная и вторичная память для хранения информации;
  + и системы связи, позволяющие различным системам взаимодействовать друг с другом.
* Программные системы управляют физическими ресурсами и преобразуют физические реализации трех абстракций в компьютерные системы, способные обрабатывать приложения.
* В компьютерных технологиях под термином "виртуализация" обычно понимается логическая 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.

Преимущества виртуализации:

1. Эффективное использование вычислительных ресурсов
2. Сокращение расходов на инфраструктуру
3. Снижение затрат на программное обеспечение.
4. Повышение гибкости и скорости реагирования системы.
5. Несовместимые приложения могут работать на одном компьютере
6. Повышение доступности приложений и обеспечение непрерывности работы предприятия
7. Возможности легкой архивации
8. Повышение управляемости инфраструктуры

В компьютерной системе имеется ряд интерфейсов, начиная с базового набора команд ЦПУ (ISA – Instruction Set Architecture) и кончая коллекцией АРI поставляемых с различным ПО современных систем с промежуточным слоем.

Например, для исполнения под ОС Unix приложений Windows, необходимо использовать эмулятор обеспечивающий преобразование системных вызовов Windows в системные вызовы UNIX.

* Виртуализация – это по сути замена существующих интерфейсов на интерфейсы имитирующие поведение других систем.

Способы виртуализации.

* Имеется несколько способов реализации виртуализации в зависимости от типа интерфейса компьютерной системы, используемого для этого:
  + Виртуализация на уровне набора команд (ISA). Используется интерфейс между аппаратными средствами и ПО, который представляет собой набор машинных команд;
  + Виртуализация на уровне системных вызовов (эмуляция системных вызовов библиотек функций ОС);
  + Виртуализация на уровне библиотечных вызовов (на уровне API) приложений.

Виртуализация ЭВМ

* Виртуализация ЭВМ — размещение нескольких логических машин в рамках одной физической.
* Цели виртуализации:
  + Предоставить каждому пользователю изолированную среду исполнения приложений.
  + Повысить гибкость использования ресурсов ЭВМ исполняемыми на ней приложениями.
  + Повысить защищенность приложений друг от друга исполняемых на одной и той же ЭВМ.
  + Повысить эффективность использования аппаратных средств ЭВМ.

Пример: Виртуализация ресурсов физического сервера

* Виртуализация ресурсов физического сервера позволяет:
  + гибко распределять их между приложениями, каждое из которых при этом "видит" только предназначенные ему ресурсы и "считает", что ему выделен отдельный сервер, т. е. в данном случае реализуется подход "один сервер — несколько приложений"

Область эффективной виртуализации

Изображение выглядит как текст, снимок экрана, круг, диаграмма

Автоматически созданное описание

* Первые версии гипервизоров отличались относительной медлительностью и действительно приводили к серьезному снижению производительности по сравнению с тем, как операционные системы и запущенные в них приложения работали на «реальном железе»
* Сегодня появилась модель предоставления виртуального хостинга на базе контейнеров ОС (или «Легких ВМ») – максимум эффективности при минимальной нагрузке на серверы.

1. История технологий виртуализации. Требования к архитектуре ЭВМ, для поддержки виртуализации.

История виртуализации (1)

* 1965. Выражение “Hypervisor” впервые появилось применительно к ПО обработки RPQ на ЭВМ IBM 360/65.
* Примерно 1966. В кембриджском научном центре разработан эмулятор CP-40 для S/360-40, явившийся первой попыткой реализации полной программной виртуализации физической ЭВМ.
* CP - Control Program или ГИПЕРВИЗОР
* 1967. На основе этой разработки была создана [CP-67/CMS](https://en.wikipedia.org/wiki/CP/CMS) – [virtual machine](https://en.wikipedia.org/wiki/Virtual_machine)/[virtual memory](https://en.wikipedia.org/wiki/Virtual_memory) [time-sharing](https://en.wikipedia.org/wiki/Time-sharing) [operating system](https://en.wikipedia.org/wiki/Operating_system) для S/360-67

История виртуализации (2)  
Первый гипервизор VM/370

* В начале 70-х гипервизор СР-67, был переработан в виде OS VM/370 для нового семейства машин System/370, выпущенного на рынок в 1972 г.
* Линейка машин System/370 в 1990-х годах была заменена компанией IBM линейкой System/390. Виртуализация в ОС MVS 390 была сохранена.
* В 2000 году IBM выпустила машины z-серии, поддерживающие 64-разрядное виртуальное адресное пространство при сохранении обратной совместимости с System/360 и поддержке виртуализации.
* Все эти системы фирмы IBM поддерживали виртуализацию на десятилетия раньше того момента, когда она приобрела популярность на машинах семейства x86.

История виртуализации (3)

* 1980-90 г.г. В это время основные работы в области виртуализации велись в направлении адаптации этой технологии для персональных ЭВМ и прежде всего для архитектуры Intel x86.
* В 1999 г. компания VMware представила технологию виртуализации систем на базе x86 получившую название VMware Virtual Platform. Первым продуктом реализующим новую технологию было ПО VMware WorkStation (гирервизор на основе хозяйской ЭВМ).
* 2005 г. VMware выпустила первое бесплатное ПО виртуализации десктопов - VMware Player.
* 2006 г. Выпустила ПО VMware ESX Server – первый гипервизор полной виртуализации серверов для архитектуры х86 (native hypervisor – “родной” гипервизор х86)

История виртуализации (4)

* Позднее в "битву“включились такие компании как:
  + Parallels (ранее SWsoft), продукты Parallels Workstation, **Parallels Desktop для Mac** (2006), **Parallels Virtuozzo Containers** (технология виртуализации средствами ОС);
  + Oracle (Sun Microsystems), **VirtualBox** (2008) была приобретена у компании Innotek в 2007;
  + Citrix Systems (**XenSourse**), Xen разработан в кембриджеском университете (там же где и CP-40), первый публичный релиз Xen выпущен в 2003. Гипервизор Xen использует технологию паравиртуализации. В октябре 2007 [Citrix](https://ru.wikipedia.org/wiki/Citrix) купила XenSource и осуществила переименование продуктов Xen;
  + RedHat. Программное обеспечение **KVM** было создано, разрабатывается и поддерживается фирмой [Qumranet](https://ru.wikipedia.org/wiki/Qumranet), которая была куплена [Red Hat](https://ru.wikipedia.org/wiki/Red_Hat) за $107 млн 4 сентября 2008 года.После сделки KVM (наряду с системой управления виртуализацией [oVirt](https://ru.wikipedia.org/wiki/OVirt)) вошла в состав платформы виртуализации [RHEV](https://ru.wikipedia.org/w/index.php?title=RHEV&action=edit&redlink=1);
  + Корпорация Microsoft вышла на рынок средств виртуализации в 2003 г. с приобретением компании Connectiх, выпустив свой первый продукт **Virtual PC** для настольных ПК.
  + Microsoft ***Hyper*-*V*** (кодовое имя Viridian), — система аппаратной виртуализации для x64-систем на основе гипервизора. Бета-версия *Hyper*-*V* была включена в x64-версии Windows Server 2008, а законченная версия (автоматически, через Windows Update) была выпущена 26 июня 2008.

История виртуализации (5)

* 2011г. Сформулированы основные модели развертывания и признаки облачных вычислений.
* В основе облачных вычислений лежат технологии виртуализации.

Требования к архитектуре ЭВМ, для поддержки виртуализации

* В 1974 году двое ученых из Калифорнийского университета (Лос-Анджелес), работающих в компьютерной сфере, Геральд Попек (Gerald Popek) и Роберт Голдберг (Robert Goldberg), опубликовали основополагающую статью («Formal Requirements for Virtualizable Third Generation Architectures»), в которой дали точный перечень тех условий, которым должна отвечать компьютерная архитектура, чтобы иметь возможность эффективно поддерживать виртуализацию.
* Такими требованиями являются:
  + 1. **Безопасность** — у гипервизора должно быть полное управление виртуализированными ресурсами.
  + 2. **Эквивалентность** — поведение программы на виртуальной машине должно быть идентичным поведению этой же программы, запущенной на реальном оборудовании.
  + 3. **Эффективность** — основная часть кода в виртуальной машине должна выполняться без вмешательства гипервизора.

1. Методы виртуализации платформы Intel х86. Требования к гипервизору. Проблемы виртуализации в архитектуре Intel x86. Два способа реализации гипервизора (монитора виртуальных машин).

Требования к гипервизору

* В 1974 г. Попек и Голдберг дали набор достаточных условий, чтобы компьютерная архитектура поддерживала виртуализацию и позволяла гипервизору работать эффективно. Их четкое описание этих условий в является важным вкладом в эту область:

1. Программа, работающая под гипервизором, должна показывать поведение, по существу идентичное тому, что было продемонстрировано при непосредственном запуске на эквивалентной машине.

2. Гипервизор должен полностью контролировать виртуализированные ресурсы.

3. Статистически значимая часть машинных инструкций должна выполняться без вмешательства гипервизора.

Проблемы виртуализации в архитектуре Intel x86 (1)

* В наборе команд Intel x86 (включая и х64) имеются команды способные изменить состояние процессора исполняемые в пользовательском режиме. Различают команды:
  + поведение которых зависит от режима исполнения (behavior sensitive) – служебные инструкции (по определению Попека и Голдберга);
  + которые влияют на управление (control senitive) – привелигерованные инструкции (по определению Попека и Голдберга);
* Например команда **РОРF**, которая устанавливает флаг разрешения прерывания, только когда работает в режиме ядра (привелигерованная инструкция), но относится к общим командам.
* Имеется еще **17** команд являющихся не привилегированными и не перехватываемых операционной системой, но чувствительных к режиму исполнения (служебные инструкции).
* Попек и Голдберг впервые утверждали что машина может быть подвергнута виртуализации, только если служебные инструкции являются поднабором привилегированных инструкций.
* В архитектуре Intel x86 при попытке сделать в пользовательском режиме то, что вы не должны делать в этом режиме, оборудование должно вызвать системное прерывание.
* В отличие от IBM/370, обладающей эти свойством, у Intel 386 его нет. При выполнении в пользовательском режиме будут проигнорированы или выполнены по-другому многие служебные инструкции 386-е машины.
* Например, инструкция POPF заменяет регистр флагов, который изменяет бит, благодаря которому блокируются и разблокируются прерывания. В пользовательском режиме этот бит просто не изменяется. Вследствие этого 386-е машины и их преемники не могли быть виртуализированы, следовательно, они не могли поддерживать гипервизор напрямую.
* Вдобавок к проблемам с инструкциями, которые, выполняясь в пользовательском режиме, вызывали системные прерывания, были еще и инструкции, которые могли считывать конфиденциальное состояние, не вызывая системных прерываний.
* Например, на процессорах x86 выпуска до 2005 года программа путем чтения селектора своего кодового сегмента могла определять, в каком режиме она выполняется, в пользовательском или в режиме ядра.
* Операционная система, выполнявшая такое действие и позволявшая обнаруживать, что она в данный момент находится в пользовательском режиме, могла на основе этой информации принимать неправильные решения.

Монитор виртуальных машин

* Монитор виртуальных машин обеспечивает интерфейс между гостевой ОС и хозяйской ОС. При этом подавляющая часть кода приложения, гостевой ОС, а также хозяйской ОС выполняется напрямую на аппаратных средствах физической машины (все общие команды и часть привилегированных).
* И лишь небольшая часть кода, которую составляют некоторые привилегированные команды перехватываются монитором виртуальных машин, модифицируются им, и затем передаются на исполнение аппаратным средствам.
* Для создания ВМ используется специальное ПО виртуализации получившее название монитор виртуальных машин (VMM – Virtual Machine Monitor).
* VMM играет роль интерфейса между аппаратными средствами физической ЭВМ и виртуальной (логической) ЭВМ. Он выделяет для ВМ ресурсы:
  + Процессор, память, устройства Ввода/Вывода, сеть.
* VMM создает иллюзию, что ВМ использует собственные аппаратные ресурсы.
* ВМ работает так, как если бы она работала на собственных реальных аппаратных средствах.
* ВМ позволяет пользователям выполнять несколько копий различных ОС на реальных аппаратных средствах одного компьютера.

Существует два основных способа реализации гипервизора (монитора виртуальных машин):

**1. Гипервизоры первого типа (Type 1 Hypervisors)**

Гипервизоры первого типа работают напрямую на аппаратном уровне, выступая в качестве базового программного слоя между аппаратным обеспечением и виртуальными машинами.

**Особенности:**

* Также называются "bare-metal" гипервизорами.
* Являются операционной системой сами по себе или тесно интегрированы с минимальной операционной системой.
* Обеспечивают высокую производительность и низкую задержку, так как не зависят от основной ОС.
* Используются в серверных и облачных системах для обеспечения изоляции, масштабируемости и производительности.

**Примеры:**

* VMware ESXi
* Microsoft Hyper-V (в режиме Server Core)
* Xen
* KVM (если работает без полноценной ОС)

**2. Гипервизоры второго типа (Type 2 Hypervisors)**

Гипервизоры второго типа работают поверх уже существующей операционной системы, как приложение.

**Особенности:**

* Также называются "hosted" гипервизорами.
* Используют ресурсы и системные вызовы основной операционной системы для управления виртуальными машинами.
* Более просты в настройке и использовании, но могут быть менее производительными из-за зависимости от основной ОС.
* Применяются на пользовательских устройствах и для тестирования, разработки или обучения.

**Примеры:**

* VMware Workstation
* Oracle VirtualBox
* Parallels Desktop
* Microsoft Hyper-V (в режиме Desktop Experience)

1. Стратегии обработки невиртуализируемых инструкций архитектуры Intel x86. Два основных подхода к виртуализации на платформе Intel x86.

Стратегии обработки невиртуализируемых инструкций

* Для обработки невиртуализируемых инструкций можно прибегнуть к двум стратегиям:
* **Двоичная трансляция.** 
  + Гипервизор отслеживает работу гостевых операционных систем;
  + Невиртуализируемые инструкции, выполняемые гостевой ОС, заменяются другими инструкциями (замена двоичных кодов проблемных команд, аналогичными, но допускающими виртуализацию)
* **Паравиртуализация.** 
  + Код гостевой ОС модифицируется для использования только инструкций, которые можно виртуализировать.
* Эти две стратегии позволяют реализовать соответствующие подходы к виртуализации.

Два основных подхода к виртуализации

* Существует два основных подхода к виртуализации процессора, :
  + ***Полная виртуализация***, когда каждая виртуальная машина работает на точной копии реального оборудования;
  + ***Паравиртуализация***, когда каждая виртуальная машина работает на слегка измененной копии реального оборудования.

1. Типы виртуальных машин: - ВМ процесс; ВМ система. Примеры ПО реализующего ВМ как процесс.

Два типа виртуальных машин

* Существует два возможных типа виртуальных машин:
  + **ВМ процесс**
  + **ВМ система**

**ВМ процесс** - это виртуальная платформа, созданная для отдельного процесса и уничтожаемая после завершения процесса.

* + Практически все операционные системы предоставляют виртуальную машину процесса для каждого из запущенных приложений (один и тот же набор команд (ISA)). Примеры:
    - **Multi program**.
    - **Binary optimizers**, пример **Wine** – ПО исполнения Windows приложений на OC FreeBSD/Linux:
  + Более интересными виртуальными машинами процессов являются те, которые поддерживают двоичные файлы, скомпилированные с использованием другого набора инструкций (ISA). Примеры:
    - **Dynamic translators,** например, эмулятор **QUEMU** (Linux/FreeBSD) – в режиме выполнения кода с набором команд отличным от набора команд физической машины., (например кода PowePC на x86\_64).
    - **HLL (Hi Level Language) VM**, (ВМ на языке высокого уровня), например, Java машина, Python машина.

**ВМ система** - виртуальная машина, которая поддерживает ОС вместе со многими пользовательскими процессами (приложениями), для одного и того же ISA:

* + Traditional VM (гипервизор I-II типа);
  + Hosted VM (VM на гипервизоре I типа);

Для различающихся ISA:

* + Whole System VM; (ВМ использует другую ISA);
  + Codesigned VM. (ВМ способные работать на нескольких ISA, например, процессор Transmeta (VLIW/x\_86!)

1. Варианты реализации монитора виртуальных машин (ВМ – система): - native (естественная или полная), hosting (поверх хозяйской ОС), hybrid (гибридная).

**Возможные варианты реализации монитора виртуальных машин (ВМ – система)**

* Гипервизор (VMM) позволяет нескольким виртуальным машинам совместно использовать систему. Возможны несколько организаций программного стека виртуализации:
  + (В) Естественная (native она же полная )- гипервизор реализуется как тонкий программный уровень, работающий непосредственно на аппаратном обеспечении хост-машины его главное преимущество – производительность. (**Гипервизор 1-го типа**)
  + (D) Хостинг - виртуальная машина работает поверх существующей ОС. (**Гипервизор 2-го типа**)
  + (С) **Гибридный** - гипервизор использует оборудование совместно с существующей ОС. Пример z/VM .

1. Гипервизор первого типа, достоинства и недостатки. Решение проблемы одновременной работы гипервизора 1-го типа и гостевой ОС в режиме ядра.

Монитор виртуальных машин исполняемый как отдельная ОС (Гипервизор первого типа)

* На аппаратные средства устанавливается специализированная ОС (гипервизор) предназначенная для виртуализации на уровне набора команд аппаратной платформы (Vmware ESX).
* Такой монитор получил название “родного” (naitive) монитора виртуальных машин.
* Достоинства данной технологии заключаются в:
  + отсутствии потребности в хозяйской ОС , ВМ устанавливаются фактически на "голое железо", а аппаратные ресурсы используются более эффективно.
* Недостатки:
  + необходимость поддержки в гипервизоре собственных драйверов внешних устройств, из-за чего возникают высокие дополнительные накладные расходы на используемые аппаратные ресурсы,
  + отсутствие учета особенностей гостевых ОС, меньшая, чем нужно, гибкость в использовании аппаратных средств

**Полная виртуализация (Native VM)  
(Гипервизор первого типа)**

* Используются **немодифицированные** экземпляры гостевых операционных систем, а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС, в роли которой выступает обычная операционная система.
* Такая технология применяется, в частности, в VMware Server (бывший GSX Server), Parallels Desktop, Parallels Server, MS Virtual Server (Hyper-V), Virtual Iron.
* Достоинства:
* - не требуется модификации гостевой ОС ;
* - «прозрачность» виртуализации.

Недостатки :

* отсутствие учета особенностей гостевых ОС,
* меньшая, чем нужно, гибкость в использовании аппаратных средств
* накладные расходы составляют 10-15%

**Проблема одновременной работы VMM (гипервизор 1-го типа) и гостевой ОС в режиме ядра (кольцо 0)**

* **VMM** требуется доступ к памяти, ЦП и устройствам ввода-вывода хоста, т.е. режим ядра.
* **Гостевая ОС** , также должна иметь доступ ко всем ресурсам, поскольку она не знает об уровне виртуализации; для этого она такжк должна работать в кольце 0, как и VMM.
* В связи с тем, что в каждый момент времени только одно ядро может работать в кольце 0 то гостевые операционные системы должны работать в другом кольце с меньшими привилегиями (**кольцо 1**) или они должны быть изменены для работы в пользовательском режиме (как при паравиртуализации).

Использование кольца 1 защиты памяти

* Многие решения по виртуализации содержали гипервизор в режиме ядра (кольцо 0), приложения — в пользовательском режиме (кольцо 3), а **гостевую операционную систему** **помещали на уровень с промежуточной привилегией (кольцо 1).**
* В результате ядро имело более высокую привилегированность по отношению к пользовательским процессам, и любая попытка доступа к памяти ядра из пользовательской программы приводила к нарушению прав доступа.
* В то же время **привилегированные инструкции гостевой операционной системы** вызывали **системное прерывание** с передачей **управления гипервизору**.
* **Гипервизор проводил ряд проверок** корректности, а затем **выполнял инструкции** от **имени гостевой** операционной системы.

1. Гипервизор второго типа, достоинства и недостатки.

**Монитор виртуальных машин исполняемый в среде хозяйской ОС  
(Гипервизор второго типа)**

* Монитор виртуальных машин работает в рамках хозяйской ОС.
* В процессе своей работы он модифицирует код гостевых ОС с целью обеспечения их виртуализации на уровне набора команд.
* При этом код содержащий “чувствительные” команды переписывается на лету (Vmware WorkStation).
* **Накладные расходы составляют 10-15%**

1. Паравиртуализация достоинства и недостатки. Управление памятью при использовании паравиртуализации.

**Паравиртуализация**

* Техника виртуализации, при которой **гостевые операционные системы подготавливаются для исполнения в виртуализированной среде**, для чего **их ядро** незначительно **модифицируется**.
* Операционная система взаимодействует с программой гипервизора, который предоставляет ей **гостевой API**, вместо использования напрямую таких ресурсов, как таблица страниц памяти.
* Этот подход не только поддерживает **высокую производительность**, но и позволяет формировать гетерогенную среду, в которой работает несколько гостевых операционных систем.
* **Накладные расходы составляют 5-10%**
* Модификация **ядра гостевой ОС** выполняется таким образом, что в нее **включается новый набор API**, через который она может **напрямую работать с аппаратурой**, не конфликтуя с другими виртуальными машинами. При этом нет необходимости задействовать полноценную ОС в качестве хостового ПО, функции которого в данном случае исполняет специальная система, получившая название гипервизора (hypervisor).
* Именно этот вариант является сегодня наиболее актуальным направлением развития серверных технологий виртуализации и применяется в VMware ESX Server, Xen (и решениях других поставщиков на базе этой технологии), Microsoft Hyper-V.

Управление памятью при полной и пара- виртуализации

* Имеются два подхода к построению мониторов виртуальных машин:
  + **Полная виртуализация (эмуляция)** всех инструкций, влечет за собой резкое **падение производительности**. Этот подход в измененном виде используется совремными **мониторами виртуальных машин**.
  + Использование **паравиртуализации**, которая предполагает **модификацию гостевых ОС**, таким образом, чтобы нейтрализовать, либо полностью исключить влияние “чувствительных” команд, средствами хозяйской ОС.

Достоинства и недостатки паравиртуализации

* Достоинства данной технологии заключаются в **отсутствии потребности в хостовой ОС** – ВМ, устанавливаются фактически на "голое железо", а **аппаратные ресурсы используются эффективно**.

Недостатки

* в сложности реализации подхода и необходимости создания специализированной ОС-гипервизора.
* **гостевые ОС необходимо модифицировать**

1. Платформа виртуализации Hyper-V.

Особенности архитектуры Hyper-v

* Hyper-v представляет собой гипервизор, т.е. прослойку между оборудованием и виртуальными машинами уровнем ниже операционной системы.



1. Платформа виртуализации VMware.

**VMware**

* В 1998 году VMware запатентовала свои программные техники виртуализации и с тех пор выпустила немало эффективных и профессиональных продуктов для виртуализации различного уровня: от VMware Workstation, предназначенного для настольных ПК, до VMware ESX Server, позволяющего консолидировать физические серверы предприятия в виртуальной инфраструктуре.
* В процессорах на базе **x86 содержатся 17 особых инструкций**, создающих проблемы при виртуализации, из-за которых операционная система отображает предупреждающее сообщение, прерывает работу приложения или просто выдает общий сбой.
* Для преодоления этого препятствия компания VMware разработала **адаптивную технологию** виртуализации, которая "**перехватывает**" данные инструкции на этапе создания и **преобразует их** в **безопасные инструкции**, пригодные для виртуализации
* **VMware Workstation** – платформа, ориентированная на desktop-пользователей и предназначенная для использования разработчиками ПО, а также проффесинальными пользователями.
* **VMware Player** – бесплатный "проигрыватель" виртуальных машин на основе виртуальной машины.
* **VMware ESX Server** – это гипервизор, разбивающий физические серверы на множество виртуальных машин. VMware ESX является основой пакета VMware vSphere и входит во все выпуски VMware vSphere.
* VMware Fusion – настольный продукт для виртуализации на платформе Mac от компании Apple.
* VMware vSphere – комплекс продуктов, представляющий надежную платформу для виртуализации ЦОД.

Структура **VMware vSphere**

* VMware vSphere включает ряд компонентов, преобразующих стандартное оборудование в общую устойчивую среду,:
* Службы инфраструктуры — это компоненты, обеспечивающие всестороннюю виртуализацию ресурсов серверов, хранилищ и сетей, их объединение и точное выделение приложениям.
* Службы приложений — это компоненты, предоставляющие встроенные элементы управления независимо от их типа или ОС.
* **VMware vCenter Server** предоставляет центральную консоль , которая поддерживает масштабируемость для управления крупными средами ЦОД.

1. Гипервизор Xen. Особенности.

Xen

* Разработка некоммерческого гипервизора Xen начиналась как исследовательский проект компьютерной лаборатории Кембриджского университета. Основателем проекта и его лидером был Иан Пратт (Ian Pratt) сотрудник университета.
* Изначально Xen представлял собой **самую развитую платформу**, поддерживающую технологию **паравиртуализации**.
* Эта технология позволяет гипервизору в хостовой системе управлять гостевой ОС посредством гипервызовов VMI (Virtual Machine Interface), что требует модификации ядра гостевой системы.
* На данный момент бесплатная версия Xen включена в дистрибутивы нескольких ОС, таких как Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. В середине августа 2007 года компания XenSource была поглощена компанией Citrix Systems.

1. Виртуализация на уровне операционной системы (контейнеризация). Достоинства и недостатки.

Виртуализация на уровне операционной системы (контейнеризация)

* Виртуализирует физический сервер на уровне ОС, позволяя запускать изолированные виртуальные серверы называемые Виртуальные Частные Серверы (Virtual Private Servers, VPS) или Контейнеры (Container, CT).
* Виртуализация на уровне операционной системы имеет минимальные накладные расходы и обеспечивает самую высокую степень консолидации, однако эта технология не позволяет запускать ОС с ядрами, отличными от ядра базовой ОС.
* **Накладные потери производительности контейнеров составляют 3%**
* **Однако степень изоляции контейнеров друг от друга ниже чем у традиционных ВМ (управляемых с помощью VMM).**

**Виртуализация на уровне ядра ОС  
(контейнеризация)**

* Этот вариант подразумевает использование одного ядра хостовой ОС для создания независимых параллельно работающих операционных сред.
* Для гостевого ПО создается только собственное сетевое и аппаратное окружение.
* Такой вариант используется в продуектах
* LXC (Linux Containers),
* Virtuozzo (для Linux и Windows),
* OpenVZ (бесплатный вариант Virtuozzo) и
* Solaris Containers.

Достоинства и недостатки виртуализации на уровне ОС

* Достоинства :
* высокая эффективность использования аппаратных ресурсов,
* низкие накладные технические расходы,
* отличная управляемость,
* минимизация расходов на приобретение лицензий.

Недостатки — реализация только однородных вычислительных сред.

1. Понятие облачных вычислений. Признаки облачных вычислений. Облачные вычисления – результат синтеза целого ряда технологий и подходов.

**Облачные вычисления** – это модель предоставления по запросу доступа к **разделяемому пулу** конфигурируемых **компьютерных ресурсов** (виртуальных машин, систем хранения данных, приложений и сервисов), которые могут быть **быстро подготовлены и предоставлены** пользователю, **с минимальными усилиям** со стороны поставщика облачных услуг.

**Облачные вычисления**

* Облачные вычисления - это вычислительная парадигма, в которой вычисления переносятся с персональных компьютеров или отдельного сервера приложений на облако компьютеров.
* Потребители облака должны заботиться только о своих вычислительных потребностях, поскольку все основные детали облачной инфраструктуры скрыты от потребителя.
* Облако - это совокупность ресурсов, которая включает хранилище, серверы, базы данных, сети, программное обеспечение и т. д.
* Таким образом, облачные вычисления, показанные на рисунке, представляют собой способ получения доступа к приложениям, инфраструктуре или платформам разработки разворачиваемых по запросу пользователя как услуги получаемой через Интернет.
* Эти приложения предоставляются с оплатой по факту использования ресурсов и доступны из веб-браузера с настольного компьютера пользователя, при он не беспокоится о том, где и на каких серверах разворачивается требуемое программное обеспечение и хранятся данные.

**Особенности облачных вычислений (1)**

* Центральную роль на рынке облачных услуг играют поставщики и потребители этих услуг.
* ИТ-услуги используются в качестве товара на открытом рынке без каких-либо технологических и юридических барьеров.
* Наиболее распространенными причинами для внедрения облачных вычислений являются:
  + Потребность в предоставляемых по требованию услуг хранения данных, вычислительной мощности, виртуальных инфраструктур для развертывания своих ИС и т.п.
  + Возможность доступа к документам через Интернет, а также их обработка с использованием сложных приложений
  + Оплата только за реально использованные ресурсы облака.

**Особенности облачных вычислений (2)**

* Облачные услуги особенно привлекательны для разработчиков ПО, у которых нет инфраструктуры или которые не могут позволить себе дальнейшее расширение своей ИТ-инфраструктуры.
* Облачная инфраструктура надежна и доступна в любое время. Вычислительные службы облака являются высоконадежными, масштабируемыми и доступными не зависимо от места нахождения пользователя.
* Данные в облаке доступны отовсюду и представляются в стандартных форматах.
* Безопасность и надежность облачных вычислений будут продолжать развиваться, за счет расширения спектра используемых методов обеспечения безопасности.

**Облачные вычисления – результат синтеза целого ряда программных технологий и подходов**



Один облачный сервер обходится дешевле, чем сервер, приобретенный и установленный самой компанией, что дает:

* + более разумное расходование электроэнергии + размещение дата-центров в регионах с более выгодными тарифами.
  + Оптимизацию использования рабочей силы (один администратор - не на 100, а на 1000 серверов).
  + более высокую безопасность и надежность (в пределе).

1. Облачные вычисления как эволюция архитектуры корпоративных приложений. Типы облаков.

**Модели развертывания облачных вычислений**

* **Private cloud** (частное облако) - инфраструктура, предназначенная для использования облачных вычислений в масштабе одной организации;
* **Community cloud** (облако сообщества) - облачная инфраструктура, которая предназначена для исключительного использования облачных вычислений определенным сообществом потребителей от организаций, которые решают общие проблемы;
* **Public cloud** (публичное облако) - инфраструктура, предназначенная для свободного использования облачных вычислений широкой публикой;
* **Hybrid cloud** (гибридное облако) - это комбинация различных облачных инфраструктур (частных, публичных или сообществ), остающихся уникальными объектами, но связанных между собой стандартизованными или частными технологиями, которые обеспечивают возможность обмена данными и приложениями.

**Типы облаков. (Модель доступа)**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Препятствия для разработки облачных приложений и сервисов.

**Препятствия для развития облачных вычислений (1)**

* Перечислим несколько наиболее очевидных препятствий:
* Доступность сервиса; что происходит, когда поставщик услуг не может выполнить поставку? Может ли такая крупная компания, как например General Motors, перенести свою ИТ-деятельность в облако и получить гарантии того, что ее деятельность не пострадает от перегрузки облака?
  + Частичным решением является заключение соглашения об уровне обслуживания (SLA).
  + Временное решение, но с отрицательными экономическими последствиями, - это избыточное выделение средств, то есть наличие достаточных ресурсов для удовлетворения наибольшего прогнозируемого спроса.
* Привязка к поставщику; Как только клиент привязан к одному поставщику облачных услуг, становится трудно перейти к другому. Усилия по стандартизации в NIST пытаются решить эту проблему.
* Конфиденциальность данных и возможность аудита; это действительно серьезная проблема.
* Узкие места в передаче данных критичны для приложений с интенсивным использованием данных. Передача 1 ТБ данных в сети 1 Мбит / с занимает 8 000 000 секунд или около 10 дней; быстрее и дешевле использовать курьерскую службу и отправлять данные, перекодированные на некоторых носителях, чем отправлять их по сети. Высокоскоростные сети решат эту проблему в будущем, например, сеть 1 Гбит / с сократит это время до 8 000 секунд, или чуть более 2 часов.
* Непредсказуемость производительности; это одно из последствий совместного использования ресурсов.
* Эластичность, возможность быстрого увеличения и уменьшения. Необходимы новые алгоритмы управления распределением ресурсов и размещением рабочих нагрузок. Автономные вычисления, основанные на самоорганизации и самоуправлении, кажутся многообещающим направлением.

1. Какие приложения не следует располагать в системах облачных вычислений.

В облачных вычислениях следует избегать размещения приложений, которые:

1. **Имеют высокие требования к производительности** — Программы, которые требуют очень быстрых откликов или интенсивного использования процессора и памяти, могут не работать должным образом в облаке, где могут быть задержки из-за сетевого взаимодействия.
2. **Обрабатывают конфиденциальные или чувствительные данные** — Несмотря на усилия по обеспечению безопасности, облачные решения могут подвергать данные риску утечек или хакерских атак, особенно если они находятся в публичных облаках.
3. **Зависят от местоположения данных** — Приложения, для которых важно хранение данных в определенной физической локации (например, из-за соблюдения нормативных требований), могут столкнуться с проблемами при использовании облачных сервисов с глобальной инфраструктурой.
4. **Не могут обеспечить высокую доступность или отказоустойчивость** — Некоторые приложения, особенно старые или не масштабируемые, могут не работать эффективно в облаке, если они не поддерживают автоматическое восстановление или балансировку нагрузки.
5. **Требуют строгих требований к лицензированию или лицензии на использование** — Некоторые программы могут быть лицензионно ограничены для использования в облачных средах, или их лицензии могут быть несовместимы с облачными сервисами.
6. **Не поддерживают распределенные вычисления** — Приложения, которые не могут быть эффективно распределены между несколькими машинами или не оптимизированы для многозадачности и параллельных вычислений, могут не быть идеальными для работы в облаке.
7. **Могут иметь проблемы с сетевыми задержками** — Приложения, требующие постоянного, быстрого обмена данными между пользователями и серверами, например, в реальном времени (например, онлайн-игры), могут испытывать проблемы с производительностью в облаке из-за сетевых задержек.

В таких случаях необходимо тщательно оценить, подходят ли облачные вычисления для конкретного приложения, или стоит использовать традиционные локальные серверы и инфраструктуру.

1. Методы обеспечения масштабируемости облачных приложений и служб в зависимости от их архитектуры.

Масштабируемость облачных приложений и служб зависит от их архитектуры и типа нагрузки. Вот основные методы, применяемые для обеспечения масштабируемости:

**1. Горизонтальное масштабирование (Horizontal Scaling)**

* **Суть:** Добавление новых серверов (виртуальных машин, контейнеров или узлов).
* **Методы:**
  + Использование контейнеров (Docker, Kubernetes) для быстрой добавки новых экземпляров.
  + Автоматическое масштабирование (Auto-scaling) на основе нагрузки.
  + Балансировка нагрузки (Load balancing) для равномерного распределения трафика.
* **Примеры архитектур:**
  + Микросервисы.
  + Бессерверные вычисления (Serverless).

**2. Вертикальное масштабирование (Vertical Scaling)**

* **Суть:** Увеличение ресурсов на одном сервере (ЦП, ОЗУ, дисковая подсистема).
* **Методы:**
  + Обновление виртуальных машин с использованием больших ресурсов.
  + Использование облачных инструментов, которые позволяют быстро изменять размер виртуальной машины (например, AWS EC2, Azure VM).
* **Примеры архитектур:**
  + Монолитные приложения.
  + Приложения с ограничениями на разделение данных.

**3. Кэширование**

* **Суть:** Снижение нагрузки на базу данных или сервер приложений путем временного хранения часто запрашиваемых данных.
* **Методы:**
  + Использование распределенных кэшей (Redis, Memcached).
  + Кэширование на уровне CDN (например, Cloudflare, AWS CloudFront).
* **Примеры архитектур:**
  + Веб-приложения с высокой интенсивностью чтения.
  + API с повторяющимися запросами.

**4. Использование микросервисной архитектуры**

* **Суть:** Декомпозиция приложения на независимые сервисы.
* **Методы:**
  + Разделение по функциональным модулям (например, учет пользователей, платежи, аналитика).
  + Независимое масштабирование отдельных микросервисов.
* **Примеры архитектур:**
  + Облачные API.
  + Мобильные и веб-приложения с разными нагрузками на компоненты.

**5. Бессерверная архитектура (Serverless)**

* **Суть:** Использование функций, которые автоматически масштабируются в зависимости от нагрузки.
* **Методы:**
  + Использование AWS Lambda, Google Cloud Functions, Azure Functions.
  + Оплата только за фактическое использование ресурсов.
* **Примеры архитектур:**
  + Реакция на события.
  + Приложения с нерегулярными нагрузками.

**6. Шардирование (Sharding) и разделение данных**

* **Суть:** Разделение базы данных или файлов на части для распределения нагрузки.
* **Методы:**
  + Шардирование базы данных (MySQL, MongoDB).
  + Использование распределенных хранилищ (Cassandra, DynamoDB).
* **Примеры архитектур:**
  + Приложения с большими объемами данных.
  + Географически распределенные приложения.

**7. Распределение очередей и событий**

* **Суть:** Асинхронная обработка задач для снижения нагрузки.
* **Методы:**
  + Использование систем очередей (RabbitMQ, Kafka, SQS).
  + Обработка фоновых задач в рабочих процессах (background jobs).
* **Примеры архитектур:**
  + Приложения с интенсивными вычислениями.
  + Приложения с потоками данных в реальном времени.

**8. Масштабируемые базы данных**

* **Суть:** Оптимизация баз данных для работы с большими объемами данных.
* **Методы:**
  + Использование облачных баз данных (AWS RDS, Google BigQuery).
  + Использование NoSQL-баз данных (MongoDB, Couchbase) для нереляционных данных.
* **Примеры архитектур:**
  + Большие хранилища данных (Data Lakes).
  + Высоконагруженные приложения.

**9. Использование Content Delivery Network (CDN)**

* **Суть:** Распределение статического контента через серверы, близкие к пользователям.
* **Методы:**
  + Подключение CDN для статических файлов, видео, изображений.
  + Глобальное распределение точек доступа.
* **Примеры архитектур:**
  + Видеостриминговые сервисы.
  + Веб-сайты с глобальной аудиторией.

**10. Оптимизация архитектуры и кода**

* **Суть:** Улучшение производительности приложения на всех уровнях.
* **Методы:**
  + Сжатие данных (gzip, Brotli).
  + Минимизация сетевых запросов.
  + Оптимизация SQL-запросов.
* **Примеры архитектур:**
  + Любые архитектуры, требующие повышения эффективности.

Эти методы часто комбинируются, чтобы достичь оптимальной масштабируемости в зависимости от требований приложения и уровня нагрузки.

1. Основные модели распределенных вычислений: Р2Р (одноранговые) вычисления; кластерные вычисления; вычисления по запросу (utility computing); грид-вычисления; облачные вычисления; туманные вычисления (fog computing); гетерогенные вычисления (jungle computing).

**Одноранговые (Р2Р)вычисления**

* P2P-вычисления - это сетевая вычислительная модель для приложений, в которой компьютеры совместно используют ресурсы и сервисы посредством прямого электронного изменения, как показано на рисунке.
* Пир - это компьютер, который ведет себя одновременно и как клиент и как сервер в модели клиент/сервер. Одноранговый компьютер может отвечать на запросы от других одноранговых узлов, передавая сообщение по сети.

**Кластерные вычисления**

* В компьютерных кластерах каждый узел настроен на выполнение одних и тех же задач, контролируемых и планируемых программным обеспечением. Компоненты кластера связаны друг с другом через быстрые локальные сети, как показано на рисунке.
* Кластерные компьютерные системы доказали свою эффективность при обработке больших объемов данных и при решении сложных предварительно распараллеленых вычислительных задач.
* Развертывание кластера увеличивает производительность и отказоустойчивость.
* Архитектура вычислительного кластера является основой инфраструктуры облачных вычислений.

**Вычисления по запросу (Utility Computing)**

* Вычисления по запросу (Utility Computing) - это модель предоставления услуг, в которой поставщик услуг предоставляет клиенту вычислительные ресурсы и управление инфраструктурой в соответствии с потребностями и взимает с них плату за конкретное использование, а не по установленной ставке.
* Достоинство этой модели заключается в низкой стоимости и отсутствии затрат на первоначальную настройку, чтобы позволить себе ресурсы компьютера.
* Эта переупаковка вычислительных услуг является основой перехода к моделям вычислений по запросу, программному обеспечению как услуге и облачным вычислениям.

Свойства вычислений по запросу:

* Многозадачность
* Мультиплексирование
* Мультарендность

**Грид-вычисления**

* Грид-система (от Power grid – электро сеть) –предполагает простоту получения услуг, путем простого подключения к грид-системе, по аналогии с тем как мы получаем электроэнергию из электрических сетей.
* На самом базовом уровне грид-вычисления - это компьютерная сеть, объединяющая ресурсы вычислительных кластеров отдельных организаций, входящих в сообщество (федерацию) грид-систем.
* Вычислительная мощность, память и хранилище данных - все это ресурсы сообщества, которые авторизованные потребители могут использовать для решения конкретных задач.
* Архитектура грид-вычислений показана на рисунке В грид-системах, также как и в кластерных системах большая программа делится (распараллеливается) на ряд независимых частей, каждая и которых выполняется на отдельном процессоре.
* Каждый процессор обрабатывает свою часть большой программы и возвращает конечный результат. Даже если один процессор выйдет из строя, результат не изменится, потому что задача будет переназначена другому процессору.
* Грид-система ведет себя как независимый объект и имеет собственное управление и администрирование.
* Грид-системы делятся на:
  + на вычислительные грид: которые в первую очередь ориентированы на интенсивные вычисления
  + грид данных: используемые для управления и контроля совместным использованием огромных объемов данных.

**Облачные вычисления**

* Облачные вычисления - это вычислительная парадигма, в которой вычисления переносятся с персональных компьютеров или отдельного сервера приложений на облако компьютеров.
* Потребители облака должны заботиться только о своих вычислительных потребностях, поскольку все основные детали облачной инфраструктуры скрыты от потребителя.
* Облако - это совокупность ресурсов, которая включает хранилище, серверы, базы данных, сети, программное обеспечение и т. д.
* Таким образом, облачные вычисления, показанные на рисунке, представляют собой способ получения доступа к приложениям, инфраструктуре или платформам разработки разворачиваемых по запросу пользователя как услуги получаемой через Интернет.
* Эти приложения предоставляются с оплатой по факту использования ресурсов и доступны из веб-браузера с настольного компьютера пользователя, при он не беспокоится о том, где и на каких серверах разворачивается требуемое программное обеспечение и хранятся данные.

**Туманные вычисления**

* Туманные вычисления - это распределенная вычислительная инфраструктура, в которой **некоторые сервисы приложений обрабатываются на интеллектуальных устройствах**, а другие - в удаленном центре обработки данных - **в облаке**.
* Основная цель туманных вычислений - **повысить эффективность вычислений** и уменьшить объем данных, которые необходимо передать в облако для хранения, обработки и анализа.
* В среде туманных вычислений, как показано на рисунке, **основная обработка** происходит в концентраторе данных на **интеллектуальном мобильном** устройстве или в сети в маршрутизаторе или другом устройстве шлюза.
* Популярность этого подхода растет благодаря Интернету вещей (IoT) и огромному количеству данных, которые генерируются датчиками. **Переносить все данные в облако для обработки и анализа неэффективно, поскольку для этого требуется значительная пропускная способность сети.**

**Вычисления на гетерогенных системах (Jungle computing)**

* Jungle computing - это сочетание разнородных, иерархических и распределенных вычислительных ресурсов.
* Jungle computing – это распределенная система, состоящая из разнообразных, распределенных и крайне неоднородных высокопроизводительных компьютерных систем, предназначенная для достижения максимальной производительности.

1. Технологические предпосылки для возникновения облачных вычислений. Особенности облачных вычислений.

**Предпосылки возникновения облачных вычислений**

* **Развитие аппаратных средств:** 
  + Рост производительности процессоров (CPU, GPU);
  + Рост пропускной способности компьютерных сетей;
  + Рост емкости оперативной и дисковой памяти.
* **Развитие технологий виртуализация:**
  + Появление ПО виртуализации: VmWare, Microsoft Hyper-V, Citrix, KVM (Linux).
  + развитие технологий контейнеризации.
* **Технологическое развитие**
  + Каналы связи
  + Интеллектуальные: сервера, сетевые устройства, устройства хранения данных
* **Сервисный подход к управлению ИТ:**
  + Технологии ITIL, SLA
* **Сервисный подход к созданию ИС:** 
  + Архитектура SOA
  + Веб-сервисы RestFull
  + микросервисы

**Особенности облачных вычислений (1)**

* Центральную роль на рынке облачных услуг играют поставщики и потребители этих услуг.
* ИТ-услуги используются в качестве товара на открытом рынке без каких-либо технологических и юридических барьеров.
* Наиболее распространенными причинами для внедрения облачных вычислений являются:
  + Потребность в предоставляемых по требованию услуг хранения данных, вычислительной мощности, виртуальных инфраструктур для развертывания своих ИС и т.п.
  + Возможность доступа к документам через Интернет, а также их обработка с использованием сложных приложений
  + Оплата только за реально использованные ресурсы облака.

**Особенности облачных вычислений (2)**

* Облачные услуги особенно привлекательны для разработчиков ПО, у которых нет инфраструктуры или которые не могут позволить себе дальнейшее расширение своей ИТ-инфраструктуры.
* Облачная инфраструктура надежна и доступна в любое время. Вычислительные службы облака являются высоконадежными, масштабируемыми и доступными не зависимо от места нахождения пользователя.
* Данные в облаке доступны отовсюду и представляются в стандартных форматах.
* Безопасность и надежность облачных вычислений будут продолжать развиваться, за счет расширения спектра используемых методов обеспечения безопасности.

1. Основные свойства (характеристики) облачных вычислений.

**ВАЖНЕЙШИЕ ХАРАКТЕРИСТИКИ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ (ПО NIST)**

Важнейшие характеристики облачных вычислений :

* Самообслуживание по запросу,
* Широкополосный доступ к сети,
* Объединение ресурсов в пулы однотипных ресурсов,
* Быструю эластичность
* Измеряемое обслуживание.

**Самообслуживание по запросу**

* Потребитель может получить доступ к вычислительным возможностям, таким как время хранения и серверное время, по мере необходимости, без какого-либо взаимодействия человека с поставщиком облачных услуг.
* Поставщики облачных услуг, предоставляющие самообслуживание по запросу, включают Google, Microsoft, Amazon Web Services (AWS), IBM и Salesforce.com.

**Широкополосный доступ к сети**

* Это означает, что размещенное в облаке приложение должно быть доступно для любого сетевого устройства (ноутбука, настольного компьютера, смартфона, планшета и т. Д.).
* Возможности облака доступны по сети и доступны через стандартные механизмы, которые способствуют использованию гетерогенных тонких или толстых клиентов.
* Потребителям просто нужен встроенный веб-браузер для подключения к поставщику облачных услуг. Это дает преимущество пользователям, у которых есть менее мощные устройства.
* Эта **мобильность** особенно привлекательна для предприятий, поскольку в рабочее или в нерабочее время пользователь может быть в курсе дел предприятия независимо от своего текущего места пребывания.

**Объединение ресурсов**

* Облако позволяет пользователям использовать данные для управления бизнесом, с помощью сериисов размещенных в облаке, одновременно, из любого места и в любое время.
* Вычислительные ресурсы объединяются для обслуживания нескольких потребителей с использованием многопользовательской модели, при этом различные физические и виртуальные ресурсы динамически назначаются и переназначаются в соответствии с потребностями потребителей.
* Пользователь обычно не знает точное местонахождение ресурсов облачного провайдера.

**Быстрая эластичность**

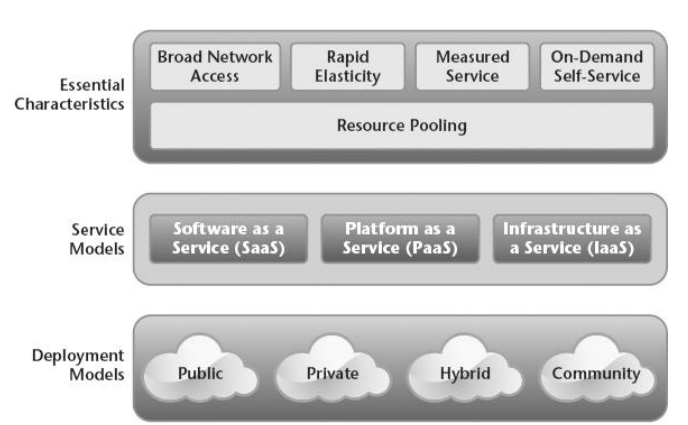
* Облако по своей природе гибкое и масштабируемое, благодаря виртуализации ресурсов.
* Потребители могут быстро и легко добавлять или удалять программные функции и другие ресурсы в соответствии с их непосредственными бизнес-потребностями.
* Облачные сервисы можно быстро и эластично предоставлять запрошенные ресурсы автоматически.

**Измеренное обслуживание**

* Это простая идея, согласно которой потребители платят только за потребляемые ресурсы.
* Облачный провайдер может измерять уровни хранения, обработки и используемой полосы пропускания, а потребителям выставляются соответствующие счета.
* Используемые ресурсы можно отслеживать и контролировать как со стороны потребителя, так и со стороны поставщика облачных услуг, что обеспечивает прозрачность.
* Сервисы облачных вычислений используют возможность измерения, которая помогает контролировать и оптимизировать использование ресурсов. Это означает, что, как и счет за электричество, ИТ-услуги также оплачиваются в соответствии с использованием - плата за использование.
* Сумма счета зависит от использования ИТ-услуг потребителями; чем больше они используют, тем выше сумма счета

1. Определение облачной системы по стандарту NIST. Модели предоставления облачных услуг (IaaS, PaaS, SaaS).

**Определение облачных вычислений по NIST**



**МОДЕЛИ ПРЕДОСТАВЛЕНИЯ ОБЛАЧНЫХ УСЛУГ**

* Общая модель облачных услуг NIST предполагает оказание поставщиками услуг, которые можно разделить на три категории, а именно:
  + программное обеспечение как услуга (Software as a Service),
  + платформа как услуга (Platform as a Srrvice),
  + инфраструктура как услуга (Infrastructure as a Service).

**Модели развертывания облаков**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Программное обеспечение как услуга (SaaS)**

* В SaaS различные типы приложений предлагаются клиенту в качестве услуги по запросу.
* В облаке запускается один экземпляр службы, и несколько конечных пользователей могут получить доступ к службам одновременно.
* Заказчикам не нужно вкладывать средства в серверы или лицензии на программное обеспечение, в то время как для провайдера затраты снижаются, поскольку необходимо размещать и поддерживать только одно приложение.
* В настоящее время SaaS предлагают такие компании, как Google, Salesforce и Microsoft.

**Программное обеспечение как сервис (SaaS)**

* **SaaS** – модель развертывания приложения, которая подразумевает предоставление приложения конечному пользователю как услуги по требованию (on demand).
* 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.

**Достоинства инедостатки модели SaaS**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Платформа как услуга (PaaS)**

* В PaaS уровень программного обеспечения или среды разработки инкапсулируется и предлагается как услуга, на основе которой могут быть построены другие услуги более высокого уровня.
* Заказчик может создавать свои собственные приложения в инфраструктуре поставщика. PaaS предоставляет вычислительную платформу и / или стек решений как услугу, часто использующую облачную инфраструктуру и поддерживающие облачные приложения.
* Разработчики менее ограничены такими ресурсами, как память и вычислительная мощность. Они могут использовать существующие навыки работы с Microsoft Visual Studio и Microsoft. NET для создания привлекательных приложений и сервисов, размещаемых в облаке.
* Созданы настраиваемые приложения и инструменты, которые повышают продуктивность разработчиков от имени всей инженерной организации. Google App Engine, Force.com и т. Д. - некоторые из популярных примеров PaaS

**Достоинства и недостатки PaaS**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Инфраструктура как услуга (IaaS)**

* IaaS предоставляет базовые вычислительные возможности и хранилище в качестве стандартизованных услуг по сети.
* Серверы, системы хранения, пространство центра обработки данных, сетевое оборудование и т. д. объединенные в пул и доступны пользователям.
* IaaS предоставляет инфраструктуру в составе: - сеть, хранилище данных , виртуальные машины и другие фундаментальные ресурсы, на которых потребитель может развертывать и запускать программное обеспечение.
* Это могут быть как приложения, так и операционные системы. Компании могут использовать IaaS для оптимизации своей инфраструктуры.
* Заказчик обычно развертывает собственное программное обеспечение в инфраструктуре, предоставляемой поставщиком IaaS. Типичными примерами являются Amazon, GoGrid, 3 Тера и т. д.

**Преимущества и проблемы IaaS**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Модели развертывания облачных систем (Private, Public, Community, Gibrid).

**Модели развертывания облачных вычислений**

* **Private cloud** (частное облако) - инфраструктура, предназначенная для использования облачных вычислений в масштабе одной организации;
* **Community cloud** (облако сообщества) - облачная инфраструктура, которая предназначена для исключительного использования облачных вычислений определенным сообществом потребителей от организаций, которые решают общие проблемы;
* **Public cloud** (публичное облако) - инфраструктура, предназначенная для свободного использования облачных вычислений широкой публикой;
* **Hybrid cloud** (гибридное облако) - это комбинация различных облачных инфраструктур (частных, публичных или сообществ), остающихся уникальными объектами, но связанных между собой стандартизованными или частными технологиями, которые обеспечивают возможность обмена данными и приложениями.
* **Типы облаков. (Модель доступа)**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Проблемы облачных вычислений связанные с обеспечением соглашения об уровне обслуживания (SLA) и управлением данными в облаке.

**ПРОБЛЕМЫ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ**

* Облачные вычисления можно рассматривать как новое явление, которое должно произвести революцию в способах использования Интернета, нужно быть очень осторожным в понимании рисков и проблем безопасности, возникающих при использовании облачных вычислений. К таким проблемам относятся:
  + Реализация соглашения об уровне обслуживания (SLA)
  + Управление данными в облаке
  + Безопасность
  + Совместимость (интероперабельность) облачных сервисов
  + Управление энергоресурсами
  + Мультиарендность ресурсов
  + Консолидация серверов
  + Надежность и доступность услуг
  + Наличие общих облачных стандартов

**Соглашения об уровне обслуживания (SLA)**

* Потребителям необходимо получить надлежащие и обещанные гарантии предоставления услуг от поставщиков. Это достигается за счет наличия соглашений об уровне обслуживания между потребителями и поставщиками.
* В зависимости от схемы приоритетов облако может минимизировать или закрыть приложение более низкого уровня.
* Большой проблемой для клиентов облачных услуг является оценка SLA поставщиков облачных услуг.
* Большинство поставщиков заключают соглашения об уровне обслуживания, чтобы защитить себя от судебных исков, предлагая клиентам минимальные гарантии.
* Есть некоторые важные вопросы, такие как **защита данных**, **перебои в работе** и **структура цен**, которые необходимо принять во внимание потребителю до подписания контракта с поставщиком.

**Управление данными в облаке**

* Данные, хранящиеся в облаке, могут быть очень большими, неструктурированными или частично структурированными.
* Поскольку поставщики услуг обычно не имеют доступа к системе физической безопасности центров обработки данных, они должны **полагаться на поставщика инфраструктуры** для обеспечения полной безопасности данных.
* Поставщик инфраструктуры должен достичь таких целей, как конфиденциальность и возможность аудита.
* **Конфиденциальность** для обеспечения безопасного доступа к данным и их передачи достигается с помощью криптографических протоколов.
* Возможность **аудита** предназначена **для проверки** того, были ли изменены настройки безопасности приложений.
* Этого можно добиться с помощью методов удаленной аттестации.

1. Проблемы облачных вычислений связанные с безопасностью и совместимостью облачных сервисов, а также с управлением энергетическими ресурсами.

**Безопасность**

* Проблемы безопасности препятствуют развитию облачных вычислений. Это например, такие проблемы безопасности, как:
  + ботнет (**Botnet** — сеть из зараженных «зомби»-компьютеров (ботов), которая чаще всего используются хакерами для организации DDoS-атак и массовой спам-рассылки);
  + Фишинг (**Фишинг** – это одна из разновидностей мошенничества в интернете с целью получения незаконного доступа к конфиденциальным данным пользователей);
  + потеря данных из-за перехвата, модификации или блокирования данных при передаче по сети, хранении или обработке в облаке.

**Совместимость (interoperability) облачных сервисов**

* Многие общедоступные облачные сети не предназначены для взаимодействия друг с другом и настроены как **закрытые системы**.
* **Отсутствие интеграции** между сетями затрудняет организациям объединение своих ИТ-систем в облаке для достижения повышения производительности и экономии средств.
* Чтобы преодолеть эту проблему, необходимо разработать **отраслевые стандарты**, которые помогут поставщикам облачных услуг разрабатывать совместимые платформы и обеспечивать переносимость данных.
* Организациям необходимо автоматически предоставлять услуги, управлять экземплярами виртуальных машин и работать как с облачными, так и с корпоративными приложениями, используя единый набор инструментов, который может работать с существующими программами и несколькими поставщиками облачных услуг.

**Управление энергетическими ресурсами**

* Около 53% общих эксплуатационных расходов центров обработки данных используется для эл.питания и охлаждения.
* Цель состоит не только в сокращении затрат на электроэнергию в центрах обработки данных, но и в соблюдении государственных постановлений и экологических стандартов.
* В последнее время большое внимание уделяется проектированию энергоэффективных центров обработки данных.
* Это может быть достигнуто за счет использования энергоэффективной аппаратной архитектуры, которая позволяет снизить скорость ЦП и отключить частичные компоненты оборудования.
* Планирование заданий с учетом энергопотребления и объединение серверов - два других способа снижения энергопотребления; даже отключение неиспользуемых машин приводит к снижению мощности.
* Ключевой задачей всех вышеупомянутых методов является достижение хорошего компромисса между энергосбережением и производительностью приложений.

1. Проблемы облачных вычислений связанные с мультиарендностью облачных ресурсов, с надежностью и доступностью услуг.

**Мультиарендность (multitenancy)**

* Когда несколько клиентов обращаются к одному и тому же оборудованию, серверам приложений и базам данных, это может повлиять на время отклика и производительность для других клиентов.
* В частности, для мультиарендности на уровне приложений ресурсы совместно используются на каждом уровне инфраструктуры и имеют серьезные проблемы с безопасностью и производительностью.

**Надежность и доступность услуг**

* Проблема надежности становится очевидной, когда поставщик облачных услуг предоставляет программное обеспечение по запросу как услугу (SaaS).
* Программное обеспечение должно иметь фактор качества надежности, чтобы пользователи могли получить к нему **доступ в любых сетевых условиях,** например, при медленных сетевых подключениях.
* Одним из случаев, выявленных из-за ненадежности программного обеспечения по запросу, является облачный сервис Apple MobileMe, который хранит и синхронизирует данные между несколькими устройствами.
* Проблема начинается, когда многие пользователи не могут получить доступ к почте и правильно синхронизировать данные.
* Чтобы избежать таких проблем, провайдеры обращаются к таким технологиям, как Google Gears, Adobe AIR и Curl, которые позволяют **облачным приложениям запускаться локально**; некоторые даже позволяют им работать при отсутствии сетевого подключения.

1. Проблемы облачных вычислений связанные со стандартизацией облачных технологий. Организации, занимающиеся стандартизацией облачных услуг.

**Общие облачные стандарты**

* В стандартизации облачных технологий и механизмов взаимодействия в настоящее время есть большие пробелы и большой потенциал для развития. Необходим **целостный подход** и согласованное определение целей в области стандартизации облака.
* Ниже перечислены некоторые стандарты:
  + **Открытый формат виртуализации (OVF)**: устанавливает транспортный механизм для перемещения виртуальных машин с одной размещенной платформы на еще один.
  + **P2301**: это руководство по профилям облачной переносимости и взаимодействия (CPIP).
  + **P2302**: это стандарт межоблачного взаимодействия и федерации (SIIF).
  + **Открытый интерфейс облачных вычислений (OCCI)**: его цель - разработать API для задач управления облаком. API обеспечивает взаимодействие между облачной реализацией IaaS.

**Организации, занимающиеся стандартизацией облачных сервисов (1)**

* к организациям, фокусирующимся на открытых стандартах облачных сервисов, относятся :
  + OpenStack Foundation,
  + Open Grid Forum и
  + Open Group;
* к отраслевым организациям стандартизации, которые организовали рабочие группы по облачным вычислениям, относятся:
  + Distributed Management Task Force (DMTF)
  + Storage Network Institute Association (SNIA).
  + Рабочая группа по управлению облаком в DMTF
  + Техническая группа по облачному хранению данных в SNIA;
  + DMTF и Техническая группа по облачному хранению данных в SNIA определили стандартные интерфейсы для облачных вычислений.

**Организации, занимающиеся стандартизацией облачных сервисов (2)**

* к организациям стандартизации информационных технологий, которые предлагают одобренные или работающие стандарты облачных вычислений, относятся:
  + National Institute of Standards and Technology (NIST)
  + OASIS (Organization for the Advancementof Structured Information Standards).
  + NIST опубликовал фактически признанное определение облачных вычислений;
  + OASIS продвигает черновые варианты стандарта облачных вычислений;
* к организациям, поддерживающим пользователей и предлагающим практические методы управления SLA, относятся:
  + TM Forum
  + Cloud Service Customer Council.

1. Уязвимости облачных систем.

**Уязвимости облачных систем (1)**

* Облака подвержены злонамеренным атакам и сбоям инфраструктуры, например сбоям питания. Такие события могут повлиять на серверы доменных имен в Интернете и предотвратить доступ к облаку или могут напрямую повлиять на облака.
* Примеры:
  + атака на Akmai 15.06.2004 вызвала сбой в работе доменного имени и серьезное отключение электроэнергии, затронувшее Google, Yahoo и многие другие сайты.
  + В мае 2009 года Google стал объектом серьезной атаки типа «отказ в обслуживании» (DoS), в результате которой на несколько дней были отключены такие службы, как Google News и Gmail.
  + Молния вызвала 29–30 июня 2012 г. длительный простой облака AWS состоящего из десяти центров обработки данных в четырех зонах доступности и расположенного в Восточном регионе США.

**Уязвимости облачных систем (2)**

* Облака подвержены ошибкам в ПО, пример:
  + ошибка в Elastic Load Balancer (ELB), который использовался для маршрутизации трафика на сервера AWS с доступной емкостью.
  + Похожая ошибка повлияла на процесс восстановления службы реляционной базы данных (RDS). Это событие выявило «скрытые» проблемы, которые возникают только при особых обстоятельствах.
* Риски стабильности из-за взаимодействующих сервисов:
  + Поставщик облачных приложений, поставщик облачного хранилища и поставщик сетей могут реализовывать разные политики, а непредсказуемые взаимодействия между балансировкой нагрузки и другими механизмами реагирования могут привести к динамической нестабильности.

1. Препятствия для развития облачных систем.

**Препятствия для развития облачных вычислений (1)**

* Перечислим несколько наиболее очевидных препятствий:
* Доступность сервиса; что происходит, когда поставщик услуг не может выполнить поставку? Может ли такая крупная компания, как например General Motors, перенести свою ИТ-деятельность в облако и получить гарантии того, что ее деятельность не пострадает от перегрузки облака?
  + Частичным решением является заключение соглашения об уровне обслуживания (SLA).
  + Временное решение, но с отрицательными экономическими последствиями, - это избыточное выделение средств, то есть наличие достаточных ресурсов для удовлетворения наибольшего прогнозируемого спроса.
* Привязка к поставщику; Как только клиент привязан к одному поставщику облачных услуг, становится трудно перейти к другому. Усилия по стандартизации в NIST пытаются решить эту проблему.
* Конфиденциальность данных и возможность аудита; это действительно серьезная проблема.

**Препятствия для развития облачных вычислений (2)**

* Узкие места в передаче данных критичны для приложений с интенсивным использованием данных. Передача 1 ТБ данных в сети 1 Мбит / с занимает 8 000 000 секунд или около 10 дней; быстрее и дешевле использовать курьерскую службу и отправлять данные, перекодированные на некоторых носителях, чем отправлять их по сети. Высокоскоростные сети решат эту проблему в будущем, например, сеть 1 Гбит / с сократит это время до 8 000 секунд, или чуть более 2 часов.
* Непредсказуемость производительности; это одно из последствий совместного использования ресурсов.
* Эластичность, возможность быстрого увеличения и уменьшения. Необходимы новые алгоритмы управления распределением ресурсов и размещением рабочих нагрузок. Автономные вычисления, основанные на самоорганизации и самоуправлении, кажутся многообещающим направлением.

1. Виртуализация процессора, памяти, устройств ввода/вывода и сети.

Виртуализация ресурсов центрального процессора

* Центральный процессор является ядром:
  + Реальной машины;
  + Виртуальной машины.
* Возможности виртуализации процессора:
  + Позволяет «гостевой ОС» полагать, что она имеет эксклюзивный контроль над ЦПУ;
  + В действительности ЦПУ разделяется между несколькими ОС.

Виртуализация памяти

* Для виртуализации памяти ВМ необходимо обеспечить несколько уровней трансляции адресов:
  + Витруальной памяти ВМ в физическую память ВМ;
  + Физической памяти ВМ (находится в виртуальной памяти реальной машины) в физическую память реальной машины

Виртуализация внешней памяти

* Физические устройства внешней памяти:
  + Жесткие диски (HDD);
  + Твердотельные диски (SSD);
  + Подключаемые разделы внешних систем хранения.
* Виртуальные диски:
  + **Файлы образов ВМ** и **файлы данных ВМ**, располагаемые на физических устройствах внешней памяти;
  + В случае «падения» ПО виртуализации, все виртуальные устройства внешней памяти теряются.

Виртуализация ввода/вывода

* Ппедполагает использование УВВ в качестве ВУ ВМ.
* Могут быть виртуализированы следующие устройства:
  + Сетевые карты;
  + Порты COM, LPT, USB
  + CD-ROM/DVD
* Может быть виртуализирован ввод/вывод на устройствах:
  + Видеоадаптер;
  + Мышь и клавиатура.

1. Виртуализация хранилищ данных. Архитектура LVM. Классификация виртуальных хранилищ.

**Виртуализация хранилищ данных**

Виртуализация хранилищ данных – это представление ресурсов хранения в абстрактном виде, как логические пространства хранения (тома) и без привязки к физическим накопителям.

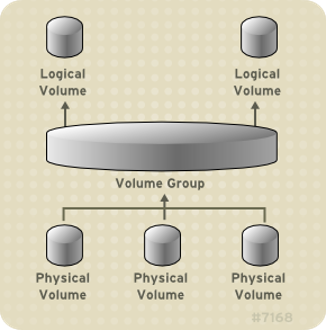
Технология обеспечивает удобный и прозрачный способ управления хранением, когда реальные устройства хранения, подключенные по различным протоколам, воспринимаются как единый пул.

**Архитектура LVM**

В основе логического тома лежит блочное устройство — раздел или целый диск. Это устройство инициализируется как физический том.

Физические тома объединяются в группы томов, тем самым создавая единое пространство для организации логических томов.

Этот процесс аналогичен разбиению дисков на разделы. Логический том будет доступен файловым системам и приложениям.



**Классификация виртуальных хранилищ**

Классификация виртуализации хранилищ описывает пять различных типов виртуализации хранилища:

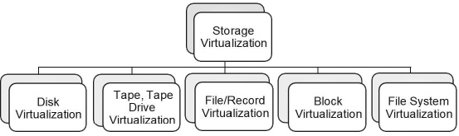
блочная,

дисковая,

лента (как носитель, накопитель и библиотека),

файловая система

виртуализация файлов



1. Блочные виртуальные хранилища данных. Достоинства и недостатки.

**Виртуализации хранилищ на основе блоков (1)**

В системе хранения на уровне блоков используются т.н. «сырые» (raw) (необработанные), т.е. не имеющие структуры тома хранения, и каждый блок можно рассматривать как отдельный жесткий диск.

Блоки контролируются серверными операционными системами и имеют возможность форматировать блоки под размещение любой файловой системы.

К преимуществам блочных систем хранения можно отнести следующее:

Предлагают лучшую производительность или скорость, чем системы хранения на уровне файлов.

Каждый блок или том хранилища может рассматриваться как независимый диск и управляется операционной системой внешнего сервера.

Блочное хранилище популярно в сетях хранения данных (SAN). Они более надежны, а их транспортные системы очень эффективны. Может поддерживать внешнюю загрузку с них операционной системы.

**Виртуализации хранилищ на основе блоков (2)**

Данные на дисках хранятся в блоках. Традиционно их размер 512 байт (один сектор).

Более крупные системы хранения, используют блоки по 4 КБ (например, 8 × 512 байтов)

Большие размеры блоков позволяют управлять или отслеживать больше данных в одном и том же месте требуя меньшего количества указателей или записей каталога.

Средствами ОС виртуальное блочное устройство может быть отформатировано под любую ФС.

Также на виртуальное блочное устройство может быть установлена любая ОС.

1. Файловые виртуальные хранилища данных. Достоинства и недостатки.

**Виртуализация хранилищ на основе файлов/записей**

Наиболее широко применяемым примером виртуализации файлов является иерархическое управление хранилищем, которое автоматизирует перенос редко используемых данных на недорогие вторичные носители, такие как оптические диски или ленточные накопители.

Эта миграция прозрачна как для пользователей, так и для приложений, которые продолжают получать доступ к данным, как если бы они находились на основном носителе.

Виртуализация файловой системы

Хранилище файлового уровня является наиболее распространенной системой хранения, которая встречается на жестких дисках, сетевых системах хранения и т. Д.

В этом типе системы хранения диск хранения настроен с использованием определенного протокола (например, NFS и т. Д.) И файлов хранятся и доступны из него оптом.

К преимуществам систем хранения на уровне файлов можно отнести следующие:

-Простота реализации

- простота использования.

1. Виртуализация сетевых хранилищ данных. Виды сетевых хранилищ (DAS, NAS, SAN).

**Виртуализация сетевых хранилищ**

Существуют три вида сетевых хранилищ данных:

хранилище с прямым подключением (DAS - Direct Attached Storage),

сетевое хранилище (NAS – Network Area Storage)

и сеть хранения данных (SAN – Storage Area Network)

**Хранилище с прямым подключением (DAS)**

Это традиционный метод хранения данных, при котором жесткие диски подключаются к физическому серверу.

Этот метод организации прост в использовании, но сложно управлять виртуализацией хранилищем этого типа.

К его недостаткам можно отнести ограниченное количество портов подключения к хранилищу (без масштабирования).

Что приводит к неэффективному использованию ресурсов (например, может ощущаться недостаточность объема используемого хранилища, при его не полном использовании.)

**Сетевое хранилище (NAS)**

Сетевое хранилище (NAS): это машина (файловый сервер), которая находится в сети и обеспечивает хранение данных для других машин.

Такой подход обеспечивает единый источник данных и упрощает резервное копирование данных.

Собирая данные в одном месте, можно также избежать проблемы доступ к нескольким серверам для доступа к данным, расположенным в разных местах.

NAS имеет следующие преимущества:

Пользователи, работающие на разных типах компьютеров (ПК, iMAC и т. д.), под управлением различных операционных систем (Windows, Unix, MacOS и т. д.), Могут обмениваться файлами.

Требуются меньшие административные расходы.

NAS - это централизованное хранилище, поэтому его легко и дешевле обслуживать, администрировать и создавать резервные копии (по сравнению с DAS).

Быстрое время отклика для пользователей, но медленнее, чем у локального диска.

К недостаткам NAS можно отнести следующее:

Оно не может предложить никаких гарантий обслуживания хранилищ для критически важных операций, поскольку NAS работает в общей среде.

NAS - это общее хранилище, поэтому оно уязвимо.

Интенсивное использование NAS может заблокировать общую локальную сеть, отрицательно сказавшись на пользователях в локальной сети.

**Сеть хранения данных (SAN)**

В этом случае развертывается специализированное оборудование (СХД – системы хранения данных) и программное обеспечение для преобразования простых дисков в высоконадежные массивы дисков для хранения данных.

СХД подключаются высокопроизводительной сети (<10Gbit) которая используется для обмена данными с машинами потребителями услуг этих СХД.

Две основные причины, по которым компании перешли на SAN:

Высокая надежность хранения. Корпоративные данные должны быть доступны круглосуточно и без выходных.

СХД имеют развитые средства управления, поэтому ими можно удобно управлять .

1. Виртуализация сети. Достоинства и недостатки виртуализации сетей. Технологии виртуализации сетевых ресурсов.

**Виртуализация сети**

Виртуализация сети - это метод объединения доступных ресурсов в сети.

Это возможно за счет разделения доступной полосы пропускания на каналы, каждый из которых независим от других, и каждый может быть назначен (или переназначен) определенному серверу или устройству в режиме реального времени.

Виртуализация сети может объединять различные физические сети в единую логическую сеть (виртуализация внешней сети) или предоставлять сетевые функции для раздела ОС (виртуализация внутренней сети).

**Достоинства и недостатки виртуализации сети**

Достоинства:

Экономит деньги за счет снижения затрат на оборудование (для виртуализации используются логические реализации сетевого оборудования средствами VMM или средствами специального ПО.

Снижает общее потребление электроэнергии.

Предоставляет возможность быстрого восстановления после аппаратного сбоя. Автоматически и мгновенно выполняет передачу с отказавшего хоста на другой, чтобы исключить простои.

Обеспечивает полное аварийное восстановление.

Помимо ряда преимуществ, виртуализация сети имеет и недостатки:

Виртуализация сети приводит к высокой степени сложности и снижению производительности.

Сетевой администратор и пользователь должны обладать высокой квалификацией;

Кроме того, виртуализация сети требует правильного и вдумчивого планирования.

В настоящее время известны следующие методы виртуализации сети: виртуальная локальная сеть (VLAN), виртуальный IP-адрес (VIP) и виртуальная частная сеть (VPN).

**Виртуальная локальная сеть (VLAN)**

Виртуальная локальная сеть (VLAN) - это метод создания независимых сетей поверх общей (физической) сети.

VLAN контролирует взаимодействие между разными сетями в одной физической сети. VLAN - это общая черта всех современных коммутаторов Ethernet.

Коммутатор Ethernet - это устройство, которое соединяет несколько сегментов сети и позволяет сетевым устройствам эффективно обмениваться данными.

Коммутатор позволяет создавать несколько виртуальных сетей, а также изолирует каждый сегмент от других. VLAN - это самый безопасный метод создания независимых логических сетей в общей (физической) сети.

Никакое взаимодействие с другими устройствами невозможно, даже если они подключены к одной и той же физической сети.

Средствами VLAN можно в общей сетевой инфраструктуре облака создать несколько специализированных виртуальных сетей, например, сети ВМ, сеть СХД, сеть связи с Интернет, сеть управления узлами виртуализации.

**Виртуальный IP-адрес (VIP)**

Это IP-адрес из частного адресного пространства, который назначается сетевому устройству и не связан с конкретным компьютером или сетевой картой (NIC). В основе механизма VIP лежит метод сетевой трансляции адресов (NAT – Network Address Translation).

Входящие пакеты отправляются на VIP, но перенаправляются (с помощью механизма NAT) на фактический сетевой интерфейс принимающего хоста или хостов.

VIP используется в технологиях виртуализации. Он поддерживает высокую доступность и балансировку нагрузки, когда у нескольких систем есть общее приложение, и они могут принимать трафик, перенаправленный сетевым устройством.

Виртуальный IP-адрес устраняет зависимость хоста от отдельных сетевых интерфейсов, и поэтому даже в случае отказа компьютера или сетевого адаптера VIP-адрес может оставаться доступным, поскольку другой сетевой адаптер может отвечать на соединение.

**Виртуальная частная сеть (VPN)**

Назначение VPN - гарантировать конфиденциальность на незащищенном сетевом канале. Обычно он используется как средство, позволяющее удаленному сотруднику подключиться к сети организации.

Обычно это делается с помощью специального программного обеспечения (например, Cisco VPN Client).

Программное обеспечение помогает в инициации соединения, и после того, как соединение установлено, все взаимодействие с ресурсами в сети происходит так, как если бы компьютер физически подключен к той же сети, но все это зависит от политик безопасности, применяемых в сети.

1. Виртуализация десктопов. Преимущества и недостатки.

**Виртуализация рабочих столов**

Виртуализация рабочего стола - это отделение рабочего стола с его операционной системой, приложениями и пользовательскими данными от базовой вычислительной системы.

Виртуализация рабочего стола предоставляет пользователям операционную среду, независимую от их локальных физических систем

Он состоит из серверов, программного обеспечения виртуализации на серверах и виртуального образа на рабочих столах.

Центр обработки данных и пользователь могут получить доступ к своей среде рабочего стола из любого рабочего места в сети.

**Виртуализация представлений (рабочих мест)**

Виртуализация представлений подразумевает эмуляцию интерфейса пользователя.

Т.е. пользователь видит приложение и работает с ним на своём терминале, хотя на самом деле приложение выполняется на удалённом сервере, а пользователю передаётся лишь картинка удалённого приложения.

В зависимости от режима работы пользователь может видеть удалённый рабочий стол и запущенное на нём приложение, либо только само окно приложения.

**Инфраструктурные проблемы рабочих мест корпоративных ИС**

С ростом масштабов организаций, использование в ИТ-инфраструктуре пользовательских ПК вызывает ряд сложностей:

большие операционные издержки на поддержку компьютерного парка;

сложность, связанная с управлением настольными ПК;

обеспечение пользователям безопасного и надежного доступа к ПО и приложениям, необходимым для работы;

техническое сопровождение пользователей;

установка и обновление лицензий на ПО и техническое обслуживание;

резервное копирование и т.д.

Решением, является применение технологии виртуализации рабочих мест сотрудников на базе инфраструктуры виртуальных ПК – Virtual Desktop Infrastructure (VDI).

**Virtual Desktop Infrastructure (VDI)**

VDI позволяет отделить пользовательское ПО от аппаратной части – персонального компьютера, - и осуществлять доступ к клиентским приложениям через терминальные устройства.

На обслуживающих серверах работает множество виртуальных машин, с такими клиентскими операционными системами, как Windows 7, Windows Vista и Windows XP или Linux операционными системами.

Пользователи дистанционно подключаются к виртуальной машине своей настольной среды. На локальных компьютерах пользователей в качестве удаленного настольного клиента могут применяться терминальные клиенты, старое оборудование с Microsoft Windows Fundamentals или дистрибутив Linux.

**Преимущества и недостатки виртуализации**   
**десктопов**

Пользователи могут получить доступ к своим настольным средам из любого места через LAN, WAN или широкополосный доступ из дома или любого другого места в любое время. Обновления приложений можно делать регулярно.

Он обеспечивает повышенную безопасность благодаря централизованному администрированию.

Это быстро и надежно, а также упрощает резервное копирование пользовательских данных.

К недостаткам виртуализации настольных компьютеров можно отнести:

Количество клиентских машин конечных пользователей, которые необходимы в сети, фиксировано.

Тонкие клиенты иногда так же дороги, как и отдельные компьютеры. Пропускной способности должно быть достаточно, чтобы избежать перегрузки в локальной сети и обеспечить хорошую обработку и просмотр приложений с рабочего стола.

Следует ограниччивать количество операционных систем, которые могут поддерживаться ПО виртуализации рабочих столов.

1. Виртуализация приложений. Достоинства и недостатки.

**Виртуализация приложений**

Подразумевает применение модели сильной изоляции прикладных программ с управляемым взаимодействием с ОС, при которой виртуализируется каждый экземпляр приложений, все его основные компоненты: файлы (включая системные), реестр, шрифты, INI-файлы, COM-объекты, службы.

Приложение исполняется без процедуры инсталляции в традиционном ее понимании и может запускаться прямо с внешних носителей (например, с флэш-карт или из сетевых папок).

Данная технология позволяет использовать на одном компьютере, а точнее в одной и той же операционной системе несколько несовместимых между собой приложений одновременно.

Виртуализация приложений позволяет пользователям запускать одно и то же заранее сконфигурированное приложение или группу приложений с сервера. При этом приложения будут работать независимо друг от друга, не внося никаких изменений в операционную систему

**Процедура виртуализации приложения**

Виртуализация приложений включает в себя выполнение следующих шагов:

i. Упаковка приложения: приложение устанавливается в специальном упаковщике, который записывает все файлы, реестры и настройки, связанные с приложением.

ii. Доставка приложения в целевую систему: упакованное приложение доставляется в целевую систему через Интернет, USB или настраиваемый механизм push.

iii. Выполнение приложения в виртуальной среде: наконец, приложение выполняется в виртуальной среде, и эта среда полностью изолирована от других приложений и базовой операционной системы.

**Преимущества и недостатки виртуализации приложений**

К преимуществам виртуализации приложений можно отнести следующее:

Нет необходимости устанавливать приложение.

Развертывание приложений происходит быстрее

Более простое и эффективное управление приложением

Использование виртуализации приложений приводит к снижению затрат

Повышенная безопасность

Единственный серьезный недостаток виртуализации приложений - постоянная поддержка полосы пропускания.

**ПО виртуализации приложений**

Фактически именно такой вариант виртуализации используется в Sun Java Virtual Machine, Microsoft Application Virtualization (ранее называлось Softgrid), Thinstall (в начале 2008 г. вошла в состав VMware), Symantec/Altiris.

**Гомогенная и гетерогенная консолидация приложений**

Сценарий гомогенной консолидации предусматривает перенос одного масштабного приложения, ранее выполнявшегося на нескольких серверах, на один, более мощный

Гетерогенная консолидация - объединению подлежат разные приложения.

1. Микросервисная архитектура в облачных приложениях.

Микросервисная архитектура

* **Множество языков, множество возможностей**
* **Децентрализованное управление данными**
* **Использование открытых стандартов типа HTTP**
* **Автоматизация инфраструктуры**
* **Проектирование под отказ (с учетом отказов отдельных микросервисов**
* **Синхронные вызовы считаются опасными**
* **Каждый сервис работает в отдельном процессе**

Понятие микросервисов

* Микросервисы противопоставляются традиционной монолитной архитектуре. Монолит означает, что компоненты продукта взаимосвязаны и взаимозависимы. Если перестает работать один - все остальные тоже «отваливаются».
* Предшественником микросервисной архитектуры является [сервис-ориентированная архитектура (SOA)](https://en.wikipedia.org/wiki/Service-oriented_architecture), которая также разделяет бизнес-логику на компоненты. По сути, микросервисная архитектура - частный случай SOA c набором более строгих правил.
* У микросервисов есть особые свойства, они же преимущества:
* **Гетерогенность**: возможность построить систему с помощью разных языков программирования и технологий;
* **Децентрализованное управление данными**: каждый микросервис содержит свой набор данных, доступный другим микросервисам только через соответствующий интерфейс;
* **Независимость инфраструктуры**: каждый микросервис - независимая единица, поэтому вносить изменения и разворачивать его можно независимо от других;
* **Масштабируемость**: чтобы увеличить производительность системы, нужно расширить только те сервисы, которые в этом нуждаются.

1. Бессерверная технология реализации облачных приложений и служб.

Бессерверная технология AWS

* Бессерверная архитектура – это способ создания и запуска приложений и сервисов без необходимости управления инфраструктурой.
* Приложение по-прежнему работает на серверах, но управление этими серверами AWS полностью берет на себя.
* Это избавляет от необходимости заниматься выделением ресурсов, масштабированием и обслуживанием серверов для запуска приложений, баз данных и систем хранения данных.
* для реализации бессерверных архитектур можно использовать облачные сервисы, такие как:
  + AWS Lambda (основан на микросервисах);
  + Amazon API Gateway;
  + Amazon DynamoDB (NoSQL-СУБД);
  + реляционная СУБД Aurora.
* Другие облака:
  + Microsoft Azure Functions
  + Google Functions
  + IBM OpenWhisk

1. Общая архитектура облачных систем и проблемы архитектурного дизайна облачных систем.

**Общее представление об облаке**

* Интернет облако представляется пользователю как массивный кластер серверов, размещенных в одном или нескольких ЦОДах.
* Эти сервера предоставляют ресурсы для выполнения веб-сервисов или распределенных приложений по запросам.
* Облачная платформа формируется динамически из серверов, программного обеспечения и ресурсов баз данных.
* Сервера в облаке могут быть как физическими, так и виртуальными машинами.
* Интерфейс пользователя обеспечивает доступ к службам обработки запросов на оказание облачных услуг.

Облачная платформа: - масштабируемая программно-аппаратная инфраструктура на основе вычислительного кластера (НРС). Включает в себя вычислительные узлы, системы хранения данных, сетевые ресурсы, программное обеспечение и базы данных.

**Цели создания платформ облачных вычислений**

* Предоставление ресурсов по запросам пользователей обеспечивая:
  + массштабируемость;
  + виртуализацию ресурсов;
  + эффективность использования ресурсов;
  + надежность предоставления ресурсов;
  + поддержку исполнения приложений Web 2.0;
  + безопасность предоставления разделяемых ресурсов и разделяемый доступ к ЦОДам.

**Технологии**

* Технологии используемые при создании облаков:
  + Платформы быстрого развертывания облачных ресурсов;
  + Создание виртуальных кластеров по запросу;
  + Обработка огромных массивов данных;
  + Методы мультиарендного учета ресурсов потребляемых пользователями одновременно;
  + Массштабируемые Web-коммуникации;
  + Рапределенные хранилища данных;
  + Службы лицензирования, определения объемов потребления ресурсов пользователями и расчета стоимости оказываемых услуг.

**Общая архитектура облаков**

* Включает следующие компоненты:
  + ЦОДы – центры обработки данных;
  + платформы снабжения виртуализированными вычислительными, сетевыми ресурсами и ресурсами СХД и СУБД из нескольких ЦОДов для обеспечения запросов от мультиарендных приложений;
  + управление снабжением ресурсами облака;
  + управление доступом к облаку;
  + каталоги предоставляемых облаком сервисов;
  + мониторинг производительности и безопасности облака.

**Многоуровневая архитектура облака**

Архитектура облака разработана как 3-х уровневая:

* Уровень инфраструктуры (IaaS);
* Уровень платформы (PaaS);
* Уровень приложений (SaaS).

Используется для описания моделей обслуживания пользователей и развертывания

**Проблемы архитектурного дизайна облаков**

1. Проблема доступности и блокировки данных.
2. Конфиденциальность данных и их защита.
3. Непредсказуемая производительность
4. Распределенные хранилища данных и широко распространенные ошибки программного обеспечения.
5. Массштабируемость, Интероперабельность и стандартизация.
6. Лицензирование ПО и разделение репутации .
7. Архитектурные принципы и свойства веб сервисов REST.

**Web-сервисы RESTful**

* Технология REST не привлекла большого внимания в 2000 году, когда Рой Филдинг (Roy Fielding) впервые представил ее в Калифорнийском университете в Ирвайне в своей диссертации "Архитектурные стили и дизайн сетевых архитектур программного обеспечения”
* В 2004 году W3C выпустил определение ещё одного стандарта под названием RESTful.
* За последние несколько лет REST стала преобладающей моделью проектирования Web-сервисов. Фактически REST оказала настолько большое влияние на Web, что практически вытеснила дизайн интерфейса, основанный на SOAP и WSDL, из-за значительного более простого стиля проектирования.
* В последние годы этот стандарт стал довольно популярным. На данный момент он используется многими известными сайтами по всему миру, всеми публичными облачными системами.

Архитектура REST основана на четырех принципах:

* Идентификация ресурса посредством URI – Ресурсы идентифицируются через URI, который предоставляет глобальное адресное пространство для поиска ресурсов и сервисов.
* Унифицированный интерфейс – GET извлекает текущее состояние ресурса в некотором представлении. POST передает новое состояние ресурса.
* Информативные сообщения – Ресурсы отделены от их представления таким образом, что их содержимое может быть доступно в различных форматах (например, HTML, XML, текст, RDF, JPEG).
* Взаимодействия через гиперссылки – Для обмена существуют различные технологии (например, переименование URI, cookies и скрытые поля формы). Состояние может быть вставлено в ответное сообщение, чтобы указать допустимое будущее состояние взаимодействия.

Базовые принципы Web-сервисов REST

* Явное использование HTTP-методов.
* Несохранение состояния.
* Предоставление URI, аналогичных структуре каталогов.
* Передача данных в XML, JavaScript Object Notation (JSON) или в обоих форматах.

**Явное использование HTTP-методов**

* Этот основной принцип проектирования REST устанавливает однозначное соответствие между операциями create, read, update и delete (CRUD) и HTTP-методами.
* Согласно этому соответствию:
  + Для создания ресурса на сервере используется POST.
  + Для извлечения ресурса используется GET.
  + Для изменения состояния ресурса или его обновления используется PUT.
  + Для удаления ресурса используется DELETE.

**Примеры правильного и неправильного использования команд HTTP**

* Использование GET не по назначению, например добавление данных в базу:
  + GET /adduser?name=Robert HTTP/1.1
* Проблема здесь в основном семантическая. Web-серверы предназначены для ответов на HTTP-запросы GET путем извлечения ресурсов согласно URI запроса (или критерию запроса) и возврата их или их представления в ответе, а не для добавления записи в базу данных.
* Использование GET по назначению, например добавление данных в базу:
* Простым способом решения этой общей проблемы является помещение имен и значений параметров URI запроса в XML-теги. Эти теги (XML-представление создаваемого объекта) можно отправить в теле HTTP-запроса POST, URI которого является родителем объекта:

Занесение: Извлечение:

POST /users HTTP/1.1 GET /users/Robert HTTP/1.1

Host: myserver Host: myserver

Content-Type: application/xml Accept: application/xml

<?xml version="1.0"?>

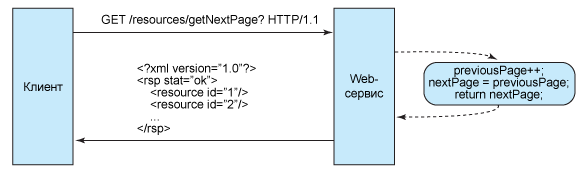
<user>

  <name>Robert</name>

</user>

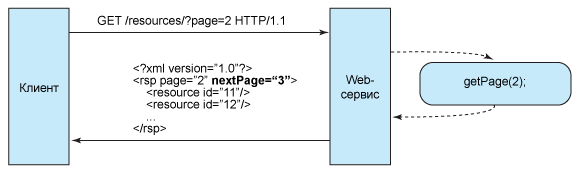
**Web сервис с сохранением состояния**

* На рисунке показан сохраняющий состояние сервис, в котором приложение может запросить следующую страницу в многостраничном наборе результатов, полагая, что сервер хранит последовательность переходов приложения по этому набору результатов.
* В этой модели с сохранением состояния сервис наращивает и сохраняет переменную previousPage



**Web-сервис без сохранения состояния**

* В Web-сервисе RESTful сервер отвечает за генерирование ответов и за предоставление интерфейса, позволяющего клиентскому приложению самому хранить свое состояние. Например, при запросе многостраничного набора результатов клиентское приложение должно включать в запрос номер конкретной страницы, а не просто запрашивать *следующую* страницу (см. рисунок 2).



**Структура URI аналогичная структуре каталогов.**

* Структура URI должна быть простой, предсказуемой и понятной.
* Один из способов достичь такого уровня удобства использования – построение URI по аналогии со структурой каталогов. Такого рода URI являются иерархическими, исходящим из одного корневого пути, ветвления которого отображают основные функции сервиса.
* Например, в сервисе обсуждений различных тем:
  + [http://www.myservice.org/discussion/topics/{topic}](http://www.myservice.org/discussion/topics/%7btopic%7d)
  + [http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}](http://www.myservice.org/discussion/%7byear%7d/%7bday%7d/%7bmonth%7d/%7btopic%7d)
  + Дополнительные рекомендации, при обдумывании структуры URI для Web-сервисов RESTful:
* Скрывайте расширения файлов серверных сценариев (.jsp, .php, .asp), чтобы можно было выполнить портирование приложений на другую технологию без изменения URI.
* Используйте только строчные буквы.
* Заменяйте пробелы дефисами или знаками подчеркивания (чем-то одним).
* Старайтесь максимально избегать использования строк запросов.
* Вместо использования кода 404 Not Found для URI, указывающих неполный путь, всегда предоставляйте в качестве ответа ресурс или страницу по умолчанию.

**Передача XML, JSON или обоих**

* Это ограничение, тесно связанный с дизайном Web-сервисов RESTful, относится к формату данных, которыми обмениваются приложение и сервис при работе в режиме запрос/ответ или в теле HTTP-запроса. Здесь особенно важны простота, читабельность и связанность.
* **Общеупотребительные MIME-типы, используемые RESTful-сервисами**

**Использование REST API  
облачными сервисами**

* На данный момент практически каждый крупный интернет-ресурс имеет свой REST API:
  + Google Drive Rest API
  + Яндекс.Диск REST API
  + Twitter REST API
  + AWS API Gateway HTTP/REST API
  + **Azure REST API**

1. Контейнеры. Контейнерная виртуализация в Linux.

**Контейнеры**

* Достоинства:
  + **Легковесность**
  + **Изоляция**
  + **Эффективность использования ресурсов**
  + **Простота мигрирования**
  + **Безопасность**
  + **Низкие накладные расходы**
  + **Одинаковые среды разработки.**
* Недостатки:
  + **Архитектура**
  + **Сложность использования ресурсоемких приложений.**

**Контейнерная виртуализация в Linux**

* Экземпляры пространств пользователя (часто называемые контейнерами или зонами) с точки зрения пользователя полностью идентичны реальному серверу, но они в своей работе используют один экземпляр ядра операционной системы.
* Для linux-систем, эта технология может рассматриваться как улучшенная реализация механизма chroot.
* Ядро обеспечивает полную изолированность контейнеров, поэтому программы из раз­ных контейнеров не могут воздействовать друг на друга.
* **Наиболее распространены сейчас OpenVZ, LXC, FreeBSD jail и Solaris Containers**

**OpenVZ**

* OpenVZ позволяет на одном физическом сервере запускать множество изолированных копий операционной системы, называемых «виртуальные частные серверы» (Virtual Private Servers, VPS) или «виртуальные среды» (Virtual Environments, VE).
* Поскольку OpenVZ базируется на ядре Linux, в отличие от виртуальных машин (напр. VMware, Parallels Desktop) или паравиртуализационных технологий (напр. Xen), в роли «гостевых» систем могут выступать только дистрибутивы Linux.
* OpenVZ является базовой платформой для Virtuozzo — проприетарного продукта Parallels

**LXC**

* **LXC** (Linux Containers) — система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров операционной системы Linux на одном узле.
* LXC основана на технологиях namespaces и cgroups, входящей в ядро Linux, начиная с версии 2.6.29.

1. Контейнеры Windows Server и Hyper-V.

**Платформа виртуализации Microsoft**

* Для Microsoft все началось, когда в 2003 году она приобрела компанию Connectix, одну из немногих компаний производящую программное обеспечение для виртуализации под Windows.
* Вместе с Connectix, компании Microsoft достался продукт Virtual PC, конкурировавший тогда с разработками компании VMware.
* В 2005 г. компания Microsoft выпустила продукт Virtual Server 2005, нацеленный на создание и консолидацию виртуальных серверов организаций.
* Летом 2008 года был выпущен финальный релиз платформы виртуализации Microsoft Hyper-V, интегрированной в ОС Windows Server 2008.

**Особенности архитектуры Hyper-v**

* Hyper-v представляет собой гипервизор, т.е. прослойку между оборудованием и виртуальными машинами уровнем ниже операционной системы. Эта архитектура была первоначально разработана IBM в 1960-е годы для мэйнфреймов и недавно стала доступной на платформах x86/x64, как часть ряда решений, включая Windows Server 2008 Hyper-V и Vmware ESX.

**Архитектура Hyper-v**



1. Использование контейнеров в Microsoft Azure.

Использование контейнеров в **Microsoft Azure** позволяет эффективно разрабатывать, развертывать и управлять приложениями в масштабируемой и изолированной среде. Azure предоставляет множество инструментов и сервисов для работы с контейнерами, что делает платформу универсальным решением для облачных и гибридных сред.

### ****1. Основные сценарии использования контейнеров в Azure****

1. **Разработка и тестирование**
   * Быстрое создание контейнеризированных приложений для разработки и тестирования.
   * Обеспечение изоляции и воспроизводимости окружений.
2. **Микросервисная архитектура**
   * Разделение приложения на независимые контейнеры, каждый из которых отвечает за свою задачу.
   * Масштабирование отдельных компонентов приложения.
3. **Переход на облачную инфраструктуру**
   * Миграция существующих приложений в контейнеры для работы в облаке.
4. **Гибридные приложения**
   * Контейнеры обеспечивают переносимость между локальными серверами и Azure.
5. **Высоконагруженные приложения**
   * Автоматическое масштабирование контейнеров в ответ на изменения нагрузки.

### ****2. Сервисы Azure для работы с контейнерами****

Microsoft Azure предоставляет широкий спектр инструментов для работы с контейнерами:

#### **1. Azure Kubernetes Service (AKS)**

* Полностью управляемый сервис Kubernetes.
* Позволяет оркестрацию контейнеров, автоматическое масштабирование и управление обновлениями.
* Подходит для развертывания микросервисных приложений.

#### **2. Azure Container Instances (ACI)**

* Легковесный сервис для развертывания одиночных контейнеров без необходимости управления инфраструктурой.
* Быстрая обработка задач, требующих временной изоляции (например, анализ данных, преобразование файлов).

#### **3. Azure App Service**

* Поддерживает развертывание контейнеризированных веб-приложений.
* Возможность интеграции с Docker Hub и частными реестрами контейнеров.

#### **4. Azure Container Registry (ACR)**

* Управляемый реестр для хранения и управления Docker-образами.
* Поддерживает безопасность, репликацию и интеграцию с AKS и ACI.

#### **5. Azure Red Hat OpenShift (ARO)**

* Управляемая платформа OpenShift для контейнеризированных приложений.
* Полезно для предприятий, использующих гибридные и мультиоблачные стратегии.

#### **6. Azure Functions с поддержкой контейнеров**

* Возможность запускать бессерверные функции в контейнерах для специализированных рабочих нагрузок.

### ****3. Преимущества контейнеров в Azure****

1. **Масштабируемость**  
   Azure позволяет автоматическое масштабирование контейнеров в зависимости от нагрузки, что снижает затраты на ресурсы.
2. **Управляемые сервисы**  
   Сервисы, такие как AKS и ACI, снимают с разработчиков необходимость настройки и обслуживания инфраструктуры.
3. **Интеграция с DevOps**  
   Контейнеры легко интегрируются с Azure DevOps и GitHub Actions для автоматизации CI/CD.
4. **Удобная экосистема**  
   Возможность интеграции с другими сервисами Azure, такими как базы данных (Azure SQL), очереди сообщений (Service Bus), мониторинг (Azure Monitor).
5. **Гибкость развертывания**  
   Поддержка гибридных решений и интеграция с локальными серверами через Azure Arc.

### ****4. Инструменты DevOps для контейнеров в Azure****

1. **Azure DevOps**
   * Инструменты CI/CD для автоматизации развертывания контейнеров в AKS или ACI.
2. **GitHub Actions**
   * Настройка CI/CD для автоматической сборки и развертывания Docker-образов в Azure.
3. **Terraform и Bicep**
   * Инфраструктура как код (IaC) для развертывания контейнерных решений.

### ****5. Примеры использования контейнеров в Azure****

1. **Развертывание микросервисного приложения в AKS**
   * Хранение Docker-образов в ACR.
   * Использование Kubernetes для управления масштабируемостью и балансировки нагрузки.
2. **Обработка больших данных в ACI**
   * Быстрое создание изолированных контейнеров для анализа данных.
3. **Веб-приложения в Azure App Service**
   * Развертывание контейнеризированного веб-приложения, интеграция с базой данных Azure SQL.
4. **Гибридные решения с Azure Arc**
   * Управление контейнерами в локальной среде и Azure через единый интерфейс.
5. **Функции в контейнерах**
   * Использование Azure Functions для специализированных вычислений, упакованных в контейнеры.

### ****6. Проблемы и вызовы****

1. **Стоимость**
   * Необходимость планирования затрат, особенно для крупных развертываний в AKS.
2. **Сложность оркестрации**
   * Kubernetes может быть сложным в настройке и управлении, даже с использованием AKS.
3. **Безопасность**
   * Необходимость управления доступом, настройкой сетевых политик и регулярного сканирования образов контейнеров.

### ****7. Заключение****

Контейнеры в Azure предоставляют мощный и гибкий инструмент для создания и управления современными приложениями. Выбор подходящего сервиса (AKS, ACI, ACR и др.) зависит от потребностей проекта и уровня сложности приложения. Благодаря глубоким возможностям интеграции с экосистемой Azure, разработчики могут эффективно использовать контейнеры как в облачных, так и в гибридных средах.

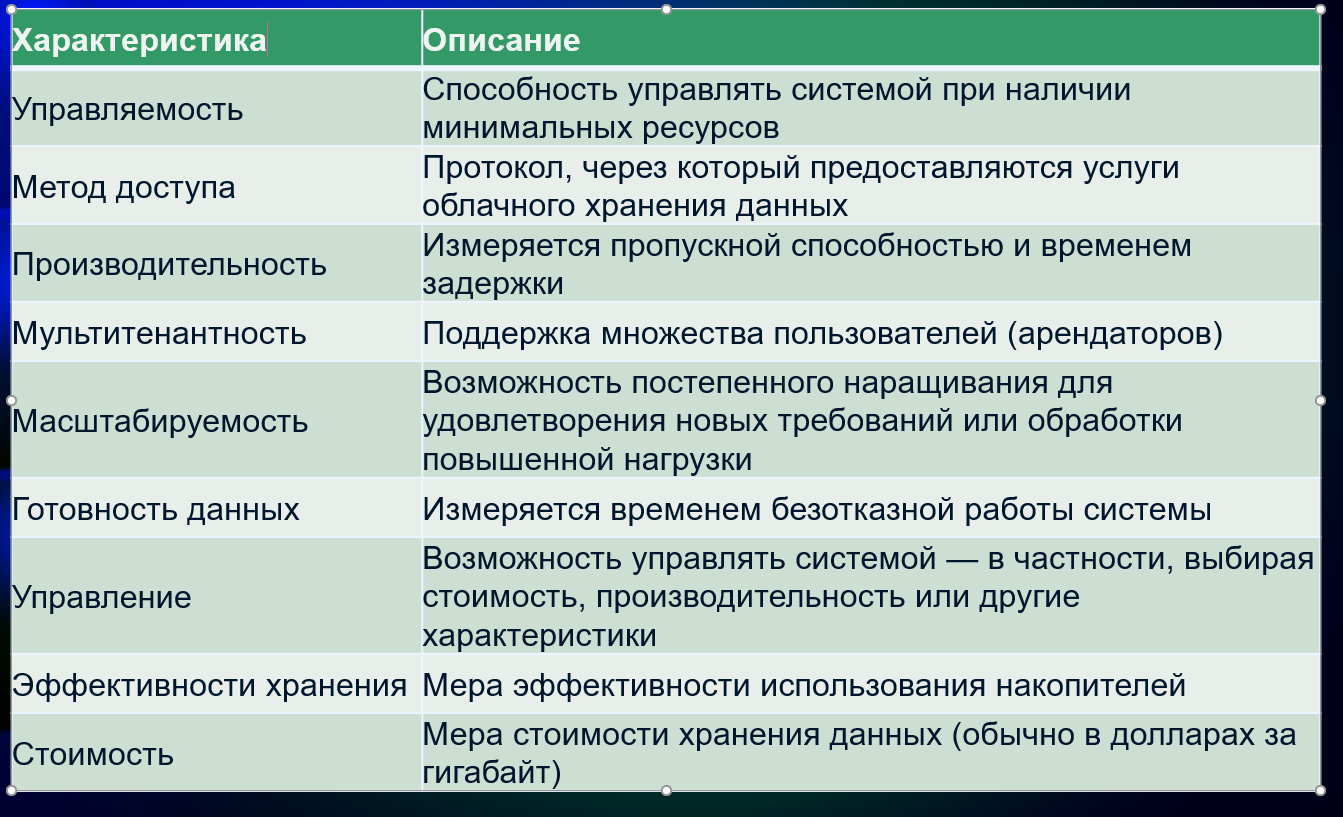
1. Облачные хранилища данных назначение, основные характеристики, преимущества и недостатки.

Облачное хранилище данных

* Облачное хранилище данных (англ. cloud storage) — модель онлайн-хранилища, в котором данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиентам.
* В отличие от модели хранения данных на собственных выделенных серверах, приобретаемых или арендуемых специально для подобных целей, количество или какая-либо внутренняя структура серверов клиенту, в общем случае, не видна.
* Данные хранятся и обрабатываются в так называемом «облаке», которое представляет собой, с точки зрения клиента, один большой виртуальный сервер.

Физически же такие серверы могут располагаться удалённо друг от друга географически.

**Характеристики о облачных систем хранения данных**



Преимущества облачных хранилищ данных (1)

1. Возможность бесплатно хранить определённый объём файлов, у каждого провайдера есть свои условия использования и ограниченное свободное место;
2. Оплата услуг идёт только за фактическое использование хранилища, а не за аренду всего сервера;
3. Уменьшение общих издержек на ИТ инфраструктуру организации, т.е. клиентам нет необходимости заниматься приобретением, поддержкой и обслуживанием собственной ресурсов по хранению данных;
4. Неограниченный доступ к данным отовсюду;

Преимущества облачных хранилищ данных (2)

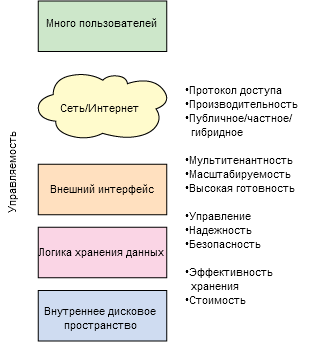
1. резервирование и все процедуры, связанные с сохранностью и целостностью данных, производятся самим провайдером облачного центра, клиент в этот процесс никак не вовлекается;
2. гибкость облачных хранилищ, сервис подходит практически, под все операционные системы, как персональные компьютеры, так и мобильные;
3. синхронизация файлов и доступ к облачным файлам, через операционную систему пользователя, зависит от самого клиента облака, поэтому нужно обращать внимание при выборе облака, на его готовность к работе с десктопным клиентом.

Недостатки ОХД

1. Если нет интернета, то нет доступа к данным в облаке.
2. Конфиденциальность информации - не все сервисы шифруют данные в облаке, что создает потенциальную угрозу перехвата информации.
3. Общая архитектура хранения данных в облаке.

**Общая архитектура**

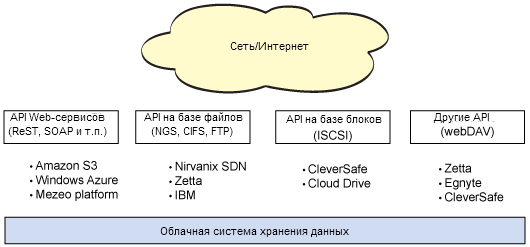
* Облачная архитектура хранения данных ― это прежде всего доставка ресурсов хранения данных по требованию в высокомасштабируемой и мультитенантной (мультиарендной) среде.
* Обобщенно облачная архитектура хранения данных представляет собой внешний интерфейс, который предоставляет API для доступа к накопителям



1. Методы доступа к облачным хранилищам данных. Облачные модели хранения данных.

**Методы доступа к ОХД**

* Основные различия между облачной и традиционной системами хранения являются средства доступа к ним.

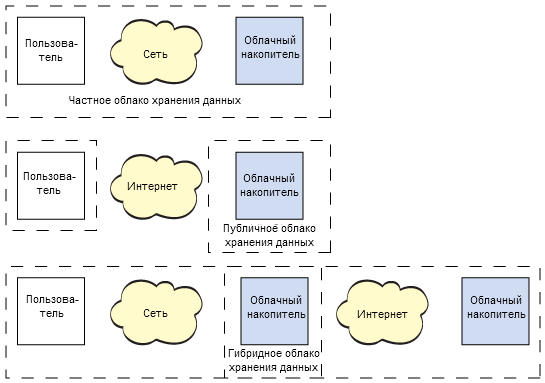


**Методы доступа (2)**

* Для доступа к ОХД используются следующие методы доступа:
  + API Web-сервисов в том числе RESTfull-API (Amazon S3), Windows Azure™ , Mezeo Cloud Storage Platform;
* Использование API Web-сервиса, требует интеграции с приложенем, поэтому используются и общие методы доступа:
  + На основе файлов:
    - NFS;
    - Common Internet File System(CIFS);
    - FTP;
  + На основе блоков:
    - iSCSI\
  + Другие:
    - WebDAV (Web-based Distributed Authoring and Versioning)

**Облачные модели хранения данных**

* Частное облако ЧД (IBM, Parascale и Cleversafe (которая предлагает программное обеспечение и/или оборудование для внутреннего облака));
* Публичное облако ХД (Amazon и Nirvanix);
* Гибридное облако ХД (Nirvanix, Egnyte).



1. Обеспечение производительности облачных хранилищ данных. Протокол FASP.

**Производительность (Проблема TCP)**

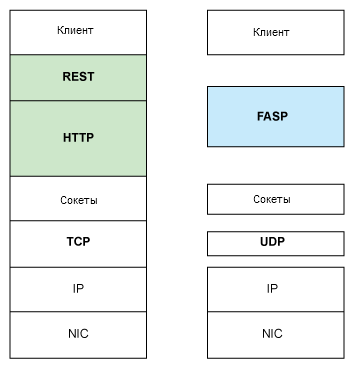
* Главная задача ОСХД ― это перемещение данных между пользователем и удаленным поставщиком облачных услуг. Проблема кроется в TCP, который управляет потоком данных на основе подтверждения приема пакетов из удаленного узла.
* Потеря или задержка пакетов приводит к применению мер по ограничению заторов пакетов с дополнительным ограничением производительности во избежание глобальных сетевых проблем.
* TCP идеально подходит для перемещения небольших объемов данных через глобальную сеть Интернет, но не для доставки больших объемов данных – в этом случае время обмена данными (RTT) увеличивается.
* Amazon с помощью Aspera Software решила эту проблему, исключив из уравнения TCP. Для ускорения массового перемещения данных во избежание больших RTT и крупных потерь пакетов разработан новый протокол *Fast and Secure Protocol* (FASP™).

**Производительность**

* Главная задача облачной системы хранения данных ― это перемещение данных между пользователем и удаленным поставщиком облачных услуг.
* Проблема кроется в том, что TCP управляет потоком данных на основе подтверждения приема пакетов из удаленного узла. Потеря или задержка пакетов приводит к применению мер по ограничению скоплений пакетов с дополнительным ограничением производительности во избежание глобальных сетевых проблем.
* Amazon с помощью Aspera Software решила эту проблему, исключив TCP при передаче данных.
* Был разработан новый протокол *Fast and Secure Protocol* (FASP™) на базе UDP. UDP позволяет управлять заторами протоколу прикладного уровня FASP

**Протокол FASP**

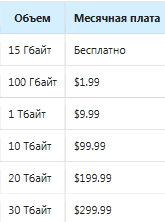
* Работая со стандартными сетевыми адаптерами (без ускорения), FASP эффективно использует доступную приложению полосу пропускания и исключает главные узкие места традиционных схем массовой передачи данных.



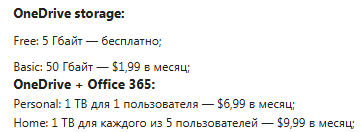
1. Обзор возможностей, преимущества и недостатки персональных облачных хранилищ: OneDrive, Dropbox, Google Drive, Яндекс Диск, Mega.

**Google Drive**

* Облачное хранилище делит пространство между Google Drive, Gmail и Google Photo.
* В сервисе можно хранить не только документы, но и фотографии, музыку, видео и многие другие файлы – всего 30 типов. Всё очень удобно и привычно для пользователей Google сервисов.
* Максимальный обьем файла 5 Тбайт.
* Доступен в веб - браузерах, Windows, Mac OS, Android, iOS и др.

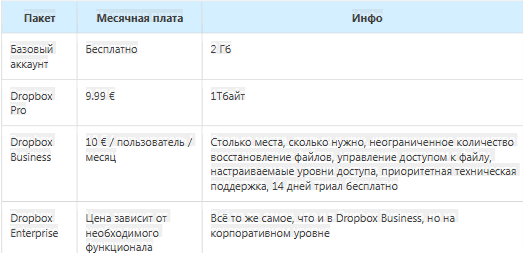


**Microsoft OneDrive**

****

* Переименованный в феврале 2014 **Microsoft SkyDrive**, базирующийся на облачной организации интернет-сервис хранения файлов с функциями файлового обмена (**SkyDrive** создан в августе 2007 года компанией Microsoft).
* Интегрирован в Office 365 и в службу **Windows Live OneDrive.**

**Dropbox**

****

* + 256 - битное шифрование AES и шифрование SSL;
  + Лучшая в своем классе технология синхронизации;
  + Интеграция с Microsoft Office 365;
  + Неограниченное восстановление файлов и журнал версий;
  + Ссылки доступа с паролем и сроком действия;
  + Настраиваемые уровни доступа;
  + Управление уровнями доступа и пр.

**Недостатки Dropbox**

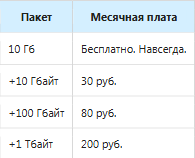
* Главным недостатком Dropbox можно считать подход к выбору папок для синхронизации. Фактически приложение следит за содержимым только одной папки Dropbox.
* Кроме того, в июле 2014 в своем интервью изданию The Guardian известный Эдвард Сноуден сделал заявление, в котором сообщил, что Dropbox не в полной мере заботится о конфиденциальности данных пользователя и даже напрямую участвует в глобальной системе слежке PRISM.
  + PRISM —государственная программа США — комплекс мероприятий, осуществляемых с целью массового негласного сбора информации, передаваемой по сетям электросвязи,
* Однако, не все согласны с этим. Использование Dropbox в связке с BoxCryptor, который надежно шифрует файлы перед синхронизацией их с облаком, обеспечивает конфидициальность данных в Dropbox.

**Mega**

****

* шифрует весь контент прямо в браузере с помощью алгоритма AES;
* пользователи могут передавать друг другу файлы в зашифрованном виде, при этом все данные хранятся в «облаке»;
* ключи доступа к файлам не публикуются в открытом доступе, а распространяются по схеме Friend
* -to-Friend, между доверяющими друг другу пользователями.
* одним из самых выгодных облачных сервисов;
* кроме того, важное отличие Mega от других подобных сервисовэто то, что он защищает личные данные пользователя.

**Яндекс.Диск**

****

* Позволяет пользователям хранить свои данные на серверах в блаке и передавать их другим пользователям в интернете.
* Работа построена на синхронизации данных между различными устройствами.
* В настоящее время регистрация пользователей доступна всем.
* Яндекс.Диск может выступать в качестве службы облачного сервиса, интегрируясь в офисный пакет Microsoft Office, а недавно появилась возможность автоматической загрузки фото и видеофайлов с цифровых камер и внешних носителей информации на Яндекс. Диск. При этом пользователю предоставляются дополнительно 32 ГБ пространства на полгода.

1. Технологии работы с большими данными в облаке. Apache Hadoop. Распределенная файловая система HDFS. Механизм репликации HDFS. [Apache Spark](http://spark.apache.org/) - альтернатива Hadoop.

**Apache Hadoop (хадуп)**

* Является одной из основополагающих технологий «[больших данных](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5)».
* Разработан на [Java](https://ru.wikipedia.org/wiki/Java) в рамках вычислительной парадигмы [MapReduce](https://ru.wikipedia.org/wiki/MapReduce), согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.
* По состоянию на 2014 год, проект состоит из четырёх модулей —
  + Hadoop Common ([связующее программное обеспечение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D1%83%D1%8E%D1%89%D0%B5%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) — набор инфраструктурных программных библиотек и утилит, используемых для других модулей и родственных проектов),
  + HDFS ([распределённая файловая система](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)),
  + YARN(система для планирования заданий и управления кластером)
  + Hadoop MapReduce(платформа программирования и выполнения распределённых MapReduce-вычислений).
* Интересный факт. *Хадуп — именно так сын одного из изобретателей фреймворка назвал желтого игрушечного слона.*

**Распределенная файловая система HDFS (1)**

* В HDFS вместо таблицы используется специальный сервер — сервер имён (NameNode), а данные разбросаны по серверам данных (DataNode).
* данные разбиты на блоки (обычно по 64Мб или 128Мб)
* HDFS имеет классическую Unix-овскую древовидную структуру директорий, пользователей с триплетом прав, и даже схожий набор консольных команд:

**Распределенная файловая система HDFS (2)**

* Выполняет 2 главные задачи:
  + запись метаданных
  + хранение собственно данных.
* Принцип работы HDFS состоит в распределении файлов по нескольким узлам в кластере.
* Так, с метаданными работает сервер NameNode, с собственно данными — DataNode.
* NameNode-сервер управляет пространством имен файловой системы и доступом клиентов к данным.
* Чтобы разгрузить NameNode-сервер, передача данных осуществляется только между клиентом и DataNode-сервером.
* HDFS является весьма надежной системой, ведь в случае уничтожения половины узлов кластера, теряется всего 3% информации.

**Механизм репликации HDFS**

* + При обнаружении NameNode-сервером отказа одного из DataNode-серверов (отсутствие heartbeat-сообщений от оного), запускается механизм репликации данных:  
    — выбор новых DataNode-серверов для новых реплик;  
    — балансировка размещения данных по DataNode-серверам.
  + Аналогичные действия производятся в случае повреждении реплик или в случае увеличения количества реплик присущих каждому блоку.

[**Apache Spark**](http://spark.apache.org/) **- альтернатива Hadoop**

* [Apache Spark](http://spark.apache.org/) представляет собой движок для распределённой обработки данных.
* Apache Spark обычно использует компоненты Hadoop, такие как HDFS и YARN для своей работы, при этом сам в последнее время стал популярнее, чем Hadoop

**Spark**

* Spark требует наличия менеджера кластера и распределенной системы хранения данных. Если вопрос управления кластерами решается путем использования нативных средств, Hadoop YARN или Apache Mesos (при работе с многоузловыми кластерами), то распределенная система хранения данных может быть исключительно посторонней.
* Именно по этой причине в большинство проектов с использованием технологий Big Data, Spark устанавливается поверх «слона»: связка из передовых приложений для аналитики от Spark и Hadoop Distributed File System позволяет программам на кластерах Hadoop выполняться до **100 раз** быстрее в RAM и до **10 раз** быстрее на ROM.
* Spark предлагает ряд операций по обработке данных. Поэтому при разработке приложений с технологиями Больших Данных, Spark чаще всего заменяет именно Hadoop MapReduce, а не весь Hadoop.

1. Технологии работы с большими данными в облаке. MapReduce.

**MapReduce**

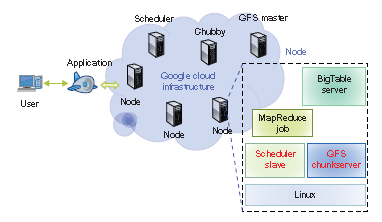
* MapReduce обеспечивает параллельные локальные вычисления на машинах кластера Hadoop с блоками данных, что обеспечивает быструю обработку больших объемов данных.
* Принцип работы MapReduce состоит из двух шагов: map и reduce. Вот как происходит процесс:
  + **Map**: получение входных данных главным узлом (master node) -> разделение информации на части -> передача данных рабочим узлам (worker node).
  + **Reduce**: получение главным узлом ответа от рабочих узлов -> формирование результата.

1. Облачная платформа Goggle – GAE. Инфраструктура и функциональная модель.

**Google App Engine (GAE)**

* Облачная платформа Google была создана по результатам экспертного анализа запросов к поисковой системе (2008).
* Google имеет несколько сотен ЦОД, развернутых по всему миру, в которых установлено более 460000 серверов (2012).
* Основной вид сервиса – PaaS, хотя имеются и облачные приложения-сервисы:
  + GMAIL, GoogleDocs, Google Earth;
  + Служба хостинга приложений;
  + Служба аккаунтов.
* Инфраструктура GAE включает в себя компоненты:
  + GFS – Google File System;
  + MapReduce – модель и сервис распределенных вычислений
  + BigTable - проприетарная высокопроизводительная база данных, построенная на основе Google File System (GFS);
  + Chubby - Бесплатный онлайн словарь;
  + Sheduler – планировщик на основе cron

**Инфраструктура GAE**



**Функциональная модель**

* Компоненты GAE обеспечивают следующую функциональность:
  + Datastore - Хранилище данных. Объектно-ориентированная, распределенная служба хранения структурированных данных на основе технологий BigTable.
  + Среда исполнения приложений - образует платформу для разработки и исполнения масшабируемых веб приложений. Языки: Java, Python.
  + SDK – инструменты разработки приложений. Позволяет тестировать приложения локально и загружать их в облако GAE4;
  + Административная консоль – используется для управления приложениями пользователей в цикле разработки, вместо использования физических ресурсов облака.
  + GAE web service infrastructure – предоставляет специальный интерфейс , обеспечивающий гибкость использования и управление ресурсами системы хранения и сетевыми ресурсами.

**Google App Engine (GAE)**

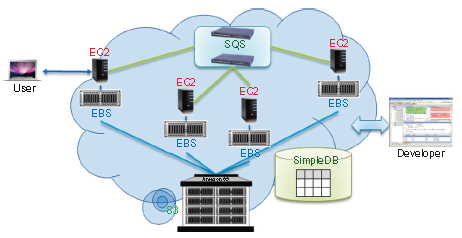
* ***Достоинства***
  + Для GAE быстро и легко разрабатывать
  + Не требует усилий по обслуживанию
  + Очень щедро выделены ежедневные бесплатные ресурсы
  + Вполне разумные цены при превышении квот
  + Лёгкое и быстрое выкладывание
* ***Недостатки***
  + Некоторые приложения невозможно реализовать из-за ограниченности функциональных возможностей
  + Контролируемое окружение не позволяет использовать некоторые стандартные библиотеки/фреймворки
  + Не поддерживаются реляционные базы данных
  + Может потребоваться дополнительная работа при миграции с GAE (если понадобится)

1. Облачная платформа Amazon Web Services - AWS. Архитектура, базовые сервисы (EC2, EBS, SSS (S3), SNS, SQS), регионы и зоны доступности.

AWS

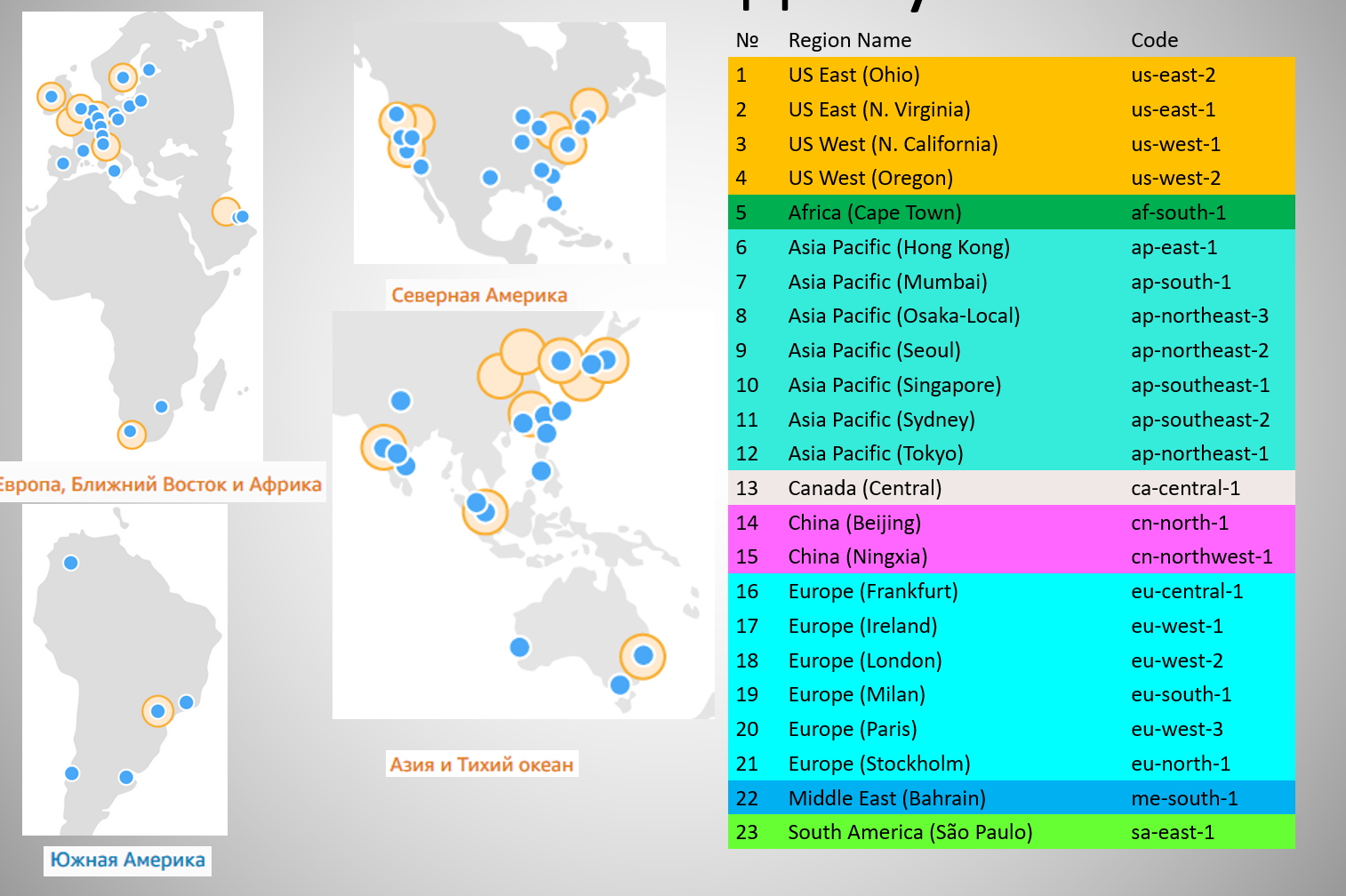
* (AWS) — инфраструктура платформ облачных веб-сервисов, представленная компанией Amazon в начале 2006 года;
* В инфраструктуре AWS представлено много сервисов для предоставления различных услуг, таких как:
  + хранение данных (файловый хостинг, распределённые хранилища данных),
  + аренда виртуальных серверов,
  + предоставление вычислительных мощностей и др.
* В 2015 г. AWS имел более миллиона активных клиентов каждый месяц из 190 стран, в том числе около 2000 правительственных учреждений, 5000 учебных заведений и более 17500 некоммерческих организаций.

Архитектура AWS



* EC2 – Elastic Compute Cloud; SQS – Simple QueueService; EBS – Elasyic block Service; S3 – Simple Storage Service; SNS – Simple Notification Service.

**Регионы и зоны доступности**

****

**Зоны доступности AWS**

* Зона доступности – это **один или несколько центров обработки данных** с резервным источником питания, сетевой конфигурацией и подключением в регионе AWS.
* Зоны доступности позволяют обеспечить для приложений и баз данных в рабочей среде **повышенный уровень доступности**, отказоустойчивость и возможность масштабирования.
* Все зоны доступности в регионе AWS объединены в полностью резервированную выделенную сеть из **оптоволокна с высокой пропускной способностью и низким уровнем задержек**, которая обеспечивает высокую скорость и низкие задержки при передаче данных между зонами доступности.
* Весь **трафик между зонами доступности зашифрован**.
* Производительность сети достаточна для синхронной репликации между зонами доступности.
* Каждая зона доступности физически расположена на большом расстоянии (много километров) от любой другой зоны доступности, хотя все они находятся в пределах 100 км (60 миль) друг от друга.

**Локальные зоны AWS**

* AWS Local Zones – это тип развертывания инфраструктуры AWS, позволяющий размещать вычислительные ресурсы, хранилища, базы данных и отдельные сервисы AWS **рядом с крупными промышленными и ИТ-центрами** с большой численностью населения.
* С помощью сервиса AWS Local Zones можно легко запускать приложения, для которых требуется **задержка менее десяти миллисекунд**, ближе к конечным пользователям в конкретном географическом регионе.
* Сервис AWS Local Zones общедоступен в **Лос-Анджелесе**. Кроме того, он доступен в **ознакомительной** версии в **Бостоне**, **Хьюстоне** и **Майами**.
* Сервис AWS Local Zones расширяет регионы AWS, находящиеся в географической близости от конечных пользователей, для запуска чувствительных к задержкам приложений с использованием таких сервисов AWS, как [Amazon Elastic Compute Cloud](https://aws.amazon.com/ru/ec2/), [Amazon Virtual Private Cloud](https://aws.amazon.com/ru/vpc/), [Amazon Elastic Block Store](https://aws.amazon.com/ru/ebs/), [Amazon FSx](https://aws.amazon.com/ru/fsx/), [Amazon Elastic Load Balancing](https://aws.amazon.com/ru/elasticloadbalancing/), [Amazon EMR](https://aws.amazon.com/ru/emr/), [Amazon ElastiCache](https://aws.amazon.com/ru/elasticache/), [Amazon Elastic Kubernetes Service](https://aws.amazon.com/ru/eks/), [Amazon Elastic Container Service](https://aws.amazon.com/ru/ecs/) и [Amazon Relational Database Service](https://aws.amazon.com/ru/rds/).

**Amazon EC2**

* **Amazon Elastic Compute Cloud (EC2)** — это облачный сервис, предоставляющий:
  + виртуальные сервера
  + 2 вида хранилищ данных:
    - краткосрочное (исчезает вместе с ВМ)
    - блочное (остается при удалении ВМ).
  + балансировщик нагрузки.

**Функциональность EC2**

* С помощью EC2 можно:
  + создать Amazon Machine Image (AMI), который будет содержать ваши приложения, библиотеки, данные и связанные с ними конфигурационные параметры. Или использовать заранее настроенные шаблоны образов для работы;
  + загрузить AMI в [Amazon S3](https://ru.wikipedia.org/wiki/Amazon_S3). Amazon EC2 предоставляет инструменты для хранения AMI. [Amazon S3](https://ru.wikipedia.org/wiki/Amazon_S3) обеспечивает безопасное, надёжное и быстрое хранилище для хранения образов;
  + использовать Amazon EC2 [Веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для настройки безопасности и сетевого доступа;
  + выбирать тип(ы) [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), какой вам необходим, запустить, завершить, или контролировать несколько AMI по мере необходимости, используя [API](https://ru.wikipedia.org/wiki/API) [Веб-сервиса](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81), или различных инструментов управления, которые предусмотрены;
  + Балансировка нагрузки и автомасштабирование. Можно создать правила при которых станет возможно автоматически увеличить количество серверов, если один или несколько серверов несправляются с нагрузкой.
  + использовать Amazon EC2 [Веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для настройки безопасности и сетевого доступа;
  + определить необходимость работать в нескольких местах, использовать статический [IP](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) или другие варианты;
  + платить только за ресурсы, которые вы собираетесь потреблять, такие как время или передача данных.

**Amazon EBS**

EBS (Elastic Block Storage) — это один из типов

хранилища используемых в инстансах (виртуальных мащинах) EC2.

• 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.

**Возможности Amazon EBS**

* **Высокопроизводительные тома**. Можно выбрать тома на основе твердотельных (SSD) или жестких (HDD) дисков, чтобы обеспечить производительность, необходимую самым требовательным приложениям.
* **Доступность.** Каждый том Amazon EBS обеспечивает проектную доступность на уровне 99,999 % и автоматически реплицируется в своей зоне доступности, что гарантирует защиту приложений на случай отказа компонентов.
* **Шифрование. Шифрование** Amazon EBS эффективно поддерживает работу с данными при хранении и передаче между инстансами EC2 и томами EBS**.**
* **Управление доступом**. Гибкие политики управления доступом Amazon позволяют определять, кому разрешен доступ к томам EBS. Такой подход обеспечивает безопасный доступ к данным.
* **Снимки состояния.** Для защиты данных используются снимки состояния томов EBS на момент времени, которые направляются на долгосрочное хранение в Amazon S3.

**Amazon S3**

* Amazon Simple Storage Service (Amazon S3) — это хранилище объектов с простым интерфейсом веб-сервисов для хранения и извлечения любого объема данных практически из любого ресурса сети.
* Amazon S3 можно использовать в качестве:
  + основного хранилища для приложений, которые изначально были предусмотрены для облака;
  + в качестве массового репозитория или «озера данных» для аналитики;
  + а также в качестве целевого объекта резервного копирования и восстановления (в том числе аварийного);
  + а также для бессерверных вычислений (AWS Lambda).

**Amazon Simple Storage Service** (Amazon S3)

* Онлайновая [веб-служба](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0), предлагаемая [Amazon Web Services](https://ru.wikipedia.org/wiki/Amazon_Web_Services), предоставляющая возможность для хранения и получения любого объёма данных, в любое время из любой точки сети, так называемый файловый [хостинг](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3).
* С помощью Amazon S3 достигается высокая масштабируемость, надёжность, высокая скорость и недорогая инфраструктура хранения данных.
* Впервые появилась в марте 2006 года в США[[1]](https://ru.wikipedia.org/wiki/Amazon_S3) и в ноябре 2007 года в Европе.[[2]](https://ru.wikipedia.org/wiki/Amazon_S3)
* Amazon S3 используется многими другими сервисами для хранения и хостинга файлов. Например, сервисы хранения и обмена файлов [Dropbox](https://ru.wikipedia.org/wiki/Dropbox) и [Ubuntu One](https://ru.wikipedia.org/wiki/Ubuntu_One), веб-сайты [Twitter](https://ru.wikipedia.org/wiki/Twitter) и [Woot.com](https://ru.wikipedia.org/w/index.php?title=Woot.com&action=edit&redlink=1), загрузчик игры [Minecraft](https://ru.wikipedia.org/wiki/Minecraft).

Amazon S3 (1)

**К бакетам S3 можно применять разного рода политики безопасности**: делать их приватными, публичными, а так же разделять права между пользователями.

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

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

* **Можно встроить BitTorrent протокол c** заменой http, как основного протокола скачивания файлов.
* **Предоставляется 99.999999999% гарантия целостности** и 99.99% гарантия доступности файлов в год (около 4-х часов недоступности в год).
* **S3 так же предполагает версионность файлов**.

Всегда можно восстановить файл предыдущей версии, т.е. откатиться до нужного состояния.

* **Пространство имен бакетов одно на всех пользователей**, поэтому названия бакетов должны быть уникальными.

**Amazon SNS**

* Amazon Simple Notification Service (Amazon SNS) — это быстрый, гибкий, полностью управляемый **сервис push-уведомлений**, который позволяет отправлять **сообщения отдельным пользователям** или рассылать их большому количеству получателей.
* Amazon SNS дает возможность без лишних сложностей и затрат отправлять push- уведомления на **мобильные устройства** пользователей, отправлять сообщения на **электронную почту** и даже в другие **распределенные сервисы**.
* Amazon SNS позволяет отправлять уведомления на устройства Apple, Google, устройства с Fire OS и Windows, а также на устройства с Android в Китае через Baidu Cloud Push. Amazon SNS можно использовать для отправки SMS пользователям мобильных устройств по всему миру.
* Кроме этих конечных точек Amazon SNS также может **передавать сообщения в Amazon Simple Queue Service** (SQS), в функции **AWS Lambda** или в **любые конечные точки HTTP**.

**Amazon Simple Queue Service (SQS)**

* **SQS** — сервис для построения очередей событий.
* Amazon SQS принимает сообщения и передает их серверам, подписанным на очередь сообщений.
* Система обмена сообщениями позволяет многим компьютерам обмениваться информацией, не имея никаких сведений друг о друге.

**Amazon Simple Queue Service** (*Amazon SQS*)

* Сервис принимает очереди сообщений для хранения.
* При использовании Amazon SQS, разработчики могут просто переместить данные, распределённые между компонентами своих приложений, которые выполняют различные задачи, не теряя при этом сообщения.
* При этом достигается высокая масштабируемость и надёжность.
* Входит в инфраструктуру сервисов [Amazon Web Services](https://ru.wikipedia.org/wiki/Amazon_Web_Services).

**Amazon SQS**

* Amazon Simple Queue Service (Amazon SQS) — это быстрый, надежный, масштабируемый и полностью управляемый сервис для **постановки сообщений в очередь**.
* Amazon SQS позволяет легко и экономично разделить компоненты облачного приложения.
* Amazon SQS можно использовать для передачи **любого объема данных** без потери сообщений и необходимости постоянного доступа к другим сервисам.
* Amazon SQS включает стандартные очереди с высокой пропускной способностью и по меньшей мере однократной обработкой и очереди **обработки данных в порядке поступления** (FIFO), обеспечивающие доставку данных по принципу FIFO и строго однократную обработку данных.

1. AWS. Сервис Вычисления, состав основных услуг. Сервис Хранилище, состав основных услуг.

Вычисления

Amazon EC2

* **Amazon Elastic Compute Cloud (EC2)** — это облачный сервис, предоставляющий:
  + виртуальные сервера
  + 2 вида хранилищ данных:
    - краткосрочное (исчезает вместе с ВМ)
    - блочное (остается при удалении ВМ).
  + балансировщик нагрузки.
* С помощью EC2 можно:
  + создать Amazon Machine Image (AMI), который будет содержать ваши приложения, библиотеки, данные и связанные с ними конфигурационные параметры. Или использовать заранее настроенные шаблоны образов для работы;
  + загрузить AMI в [Amazon S3](https://ru.wikipedia.org/wiki/Amazon_S3). Amazon EC2 предоставляет инструменты для хранения AMI. [Amazon S3](https://ru.wikipedia.org/wiki/Amazon_S3) обеспечивает безопасное, надёжное и быстрое хранилище для хранения образов;
  + использовать Amazon EC2 [Веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для настройки безопасности и сетевого доступа;
  + выбирать тип(ы) [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), какой вам необходим, запустить, завершить, или контролировать несколько AMI по мере необходимости, используя [API](https://ru.wikipedia.org/wiki/API) [Веб-сервиса](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81), или различных инструментов управления, которые предусмотрены;
  + Балансировка нагрузки и автомасштабирование. Можно создать правила при которых станет возможно автоматически увеличить количество серверов, если один или несколько серверов несправляются с нагрузкой.
  + использовать Amazon EC2 [Веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для настройки безопасности и сетевого доступа;
  + определить необходимость работать в нескольких местах, использовать статический [IP](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) или другие варианты;
  + платить только за ресурсы, которые вы собираетесь потреблять, такие как время или передача данных.

Amazon EC2 Container Service (ECS)

* Amazon EC2 Container Service (ECS) — это в высшей степени масштабируемый, высокопроизводительный **сервис для управления контейнерами**, который поддерживает контейнеры Docker.
* Устраняет необходимость в установке, эксплуатации и масштабировании собственной инфраструктуры управления кластерами.
* Простыми API-запросами можно запускать и останавливать приложения с поддержкой контейнеров Docker, запрашивать полное состояние кластера и получать доступ ко многим знакомым функциям, таким как группы безопасности, Elastic Load Balancing, тома Amazon Elastic Block Store (Amazon EBS) и роли AWS Identity and Access Management (IAM).
* Используя сервис Amazon ECS, можно запланировать размещение контейнеров в вашем кластере с учетом потребности в ресурсах и требований к доступности.
* Кроме того, для удовлетворения требований бизнеса и приложений в сервис можно интегрировать собственный или сторонний планировщик.

Amazon EC2 Container Registry (ECR)

* Amazon Elastic Container Registry (ECR) – это полностью **управляемый реестр контейнеров**, который упрощает хранение, управление, публикацию и развертывание образов контейнеров и артефактов в любом месте.
* При работе с сервисом Amazon ECR вам не придется самостоятельно управлять своими репозиториями контейнеров или заботиться о масштабировании базовой инфраструктуры.
* Amazon ECR размещает образы в высокодоступной и высокопроизводительной архитектуре, обеспечивая надежное развертывание образов для приложений на основе контейнеров.
* Amazon ECR интегрируется с сервисами [Amazon Elastic Kubernetes Service (EKS)](https://aws.amazon.com/ru/eks/), [Amazon Elastic Container Service (ECS)](https://aws.amazon.com/ru/ecs/) и [AWS Lambda](https://aws.amazon.com/ru/lambda/), что упрощает процесс разработки и выпуска рабочих версий, и с [AWS Fargate](https://aws.amazon.com/ru/fargate/) для быстрого развертывания.

Amazon Lightsail

* [Amazon Lightsail пр](https://amazonlightsail.com/)едназначен для максимально удобного запуска виртуального частного сервера и управления им с помощью AWS.21
* Планы Lightsail предоставляют все, что нужно для успешного и быстрого запуска вашего проекта:
  + виртуальную машину,
  + хранилище на основе твердотельных дисков,
  + каналы передачи данных,
  + инструменты управления DNS и статический IP-адрес,
* благодаря чему обеспечивается низкая прогнозируемая стоимость сервиса.

AWS Batch

* AWS Batch выполняет в облаке сотни тысяч пакетных вычислительных заданий для разработчиков, ученых и инженеров – эффективно и просто.
* Сервис AWS Batch предназначен для динамической подготовки вычислительных ресурсов нужного типа в оптимальном объеме (например, инстансов с оптимизацией ЦП или памяти) на основе объема отправляемых пакетных заданий и конкретных требований к ресурсам.
* При работе с AWS Batch не нужно ни устанавливать программы для пакетных вычислений, ни кластеры серверов для обработки заданий – вместо этого можно сфокусироваться на анализе результатов и решении проблем.
* AWS Batch осуществляет планирование и выполнение пакетных вычислительных рабочих нагрузок с использованием всего спектра вычислительных сервисов и возможностей платформы AWS, например сервиса Amazon EC2 и спотовых инстансов.

AWS Elastic Beanstalk

* [AWS Elastic Beanstalk —](http://aws.amazon.com/elasticbeanstalk/) это удобный сервис **для развертывания и масштабирования интернет-приложений и сервисов**, разработанных на языках Java, .NET, PHP, Node.js, Python, Ruby, Go и Docker и знакомых серверах, таких как Apache, Nginx, Passenger и IIS.23
* Можно просто загрузить свой код, и AWS Elastic Beanstalk **автоматически выполнит развертывание системы** — от подготовки ресурсов, балансировки нагрузки и автоматического масштабирования до мониторинга состояния приложения.
* При этом пользователь сохраняет полный контроль над ресурсами AWS, используемыми для приложения, и в любое время может получить к ним доступ.
* Обеспечивает гармоническое сочетание услуг , предлагаемых от Amazon Web Services **для развертывания инфраструктуры** , которая организует различные услуги AWS, в том числе EC2 , S3 , Simple Notification Service (SNS), CloudWatch , AutoScaling и Эластичная балансировка нагрузки. (EBL)
* обеспечивает дополнительный уровень абстракции над ВМ и ОС; пользователи вместо того, чтобы видеть сочетание ОС и платформы, такие как "64bit Amazon Linux 2014,03 v1.1.0 работает Рубин 2.0 (Puma)" или "64 - битной Debian Jessie v2.0.7 работает Python 3.4 (предварительно сконфигурированной - Докер)".
* Пользователь видит:
  + 'приложение' как логический контейнер для проекта,
  + A 'версии' , которая является развертываемой сборки приложения исполняемый файл,
  + 'шаблон конфигурации " , которая содержит информацию о конфигурации для окружающей среды и для продукта.
* Наконец "окружающая среда" сочетает в себе 'версию' с конфигурации и развертывает их. Исполняемые сами загружаются в виде архивных файлов S3 заранее и 'версия' это просто указатель на это.

Бессерверная технология AWS

* **Бессерверная архитектура** – это способ создания и запуска приложений и сервисов **без необходимости управления инфраструктурой**.
* Приложение по-прежнему работает на серверах, но управление этими серверами AWS полностью берет на себя.
* Это **избавляет от необходимости заниматься выделением ресурсов**, масштабированием и обслуживанием серверов для запуска приложений, баз данных и систем хранения данных.
* Для реализации бессерверных архитектур можно использовать облачные сервисы, такие как:
  + AWS Lambda (основан на микросервисах);
  + Amazon API Gateway;
  + Amazon DynamoDB (NoSQL-СУБД);
  + реляционная СУБД Aurora.
* Другие облака:
  + Microsoft Azure Functions
  + Google Functions
  + IBM OpenWhisk

AWS Lambda

* AWS Lambda позволяет запускать программные коды **без выделения серверов и управления ими**.
* Сервис Lambda позволяет выполнять код практически для любых типов приложений и серверных служб и требует практически нулевых усилий по администрированию.
* Достаточно загрузить свой код, и Lambda сделает все необходимое для запуска, масштабирования и обеспечения высокой доступности вашего кода.
* Можно настроит автоматический запуск кода из других сервисов AWS или вызывать его непосредственно из любого интернет-приложения или мобильного приложения.
* Вы оплачивается только израсходованное вычислительное время, а периоды, когда код не выполняется, не тарифицируются.

Auto Scaling

* Auto Scaling помогает поддерживать доступность приложений и масштабировать ресурсы Amazon EC2 в обоих направлениях автоматически в соответствии с определенными вами условиями.
* Функция Auto Scaling позволяет убедиться, что в системе работает нужное число экземпляров Amazon EC2.
* Auto Scaling может автоматически увеличивать количество инстансов Amazon EC2 во время пиков спроса, чтобы сохранить производительность на прежнем уровне, или понижать объемы вычислительных ресурсов во время снижения спроса, чтобы снизить расходы.
* Функция Auto Scaling отлично подходит для работы с приложениями со стабильным уровнем спроса на ресурсы и приложениями со скачками спроса в течении часа, дня или недели.

**Хранилище**

**Amazon S3**

* Amazon Simple Storage Service (Amazon S3) — это хранилище объектов с простым интерфейсом веб-сервисов для хранения и извлечения любого объема данных практически из любого ресурса сети.
* Amazon S3 можно использовать в качестве:
  + основного хранилища для приложений, которые изначально были предусмотрены для облака;
  + в качестве массового репозитория или «озера данных» для аналитики;
  + а также в качестве целевого объекта резервного копирования и восстановления (в том числе аварийного);
  + а также для бессерверных вычислений (AWS Lambda).
* Онлайновая [веб-служба](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0), предлагаемая [Amazon Web Services](https://ru.wikipedia.org/wiki/Amazon_Web_Services), предоставляющая возможность для хранения и получения любого объёма данных, в любое время из любой точки сети, так называемый файловый [хостинг](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3).
* С помощью Amazon S3 достигается высокая масштабируемость, надёжность, высокая скорость и недорогая инфраструктура хранения данных.
* Впервые появилась в марте 2006 года в США[[1]](https://ru.wikipedia.org/wiki/Amazon_S3) и в ноябре 2007 года в Европе.[[2]](https://ru.wikipedia.org/wiki/Amazon_S3)
* Amazon S3 используется многими другими сервисами для хранения и хостинга файлов. Например, сервисы хранения и обмена файлов [Dropbox](https://ru.wikipedia.org/wiki/Dropbox) и [Ubuntu One](https://ru.wikipedia.org/wiki/Ubuntu_One), веб-сайты [Twitter](https://ru.wikipedia.org/wiki/Twitter) и [Woot.com](https://ru.wikipedia.org/w/index.php?title=Woot.com&action=edit&redlink=1), загрузчик игры [Minecraft](https://ru.wikipedia.org/wiki/Minecraft).
* **К бакетам S3 можно применять разного рода политики безопасности**: делать их приватными, публичными, а так же разделять права между пользователями.
* • **S3 может логгировать запросы и складывать отчёты в отдельный бакет**. Это удобно при расследовании, когда много пользователей или приложений имеют доступ к сервису.
* • **Загрузка, удаление и другие операции доступны по REST или SOAP**, так же возможно шифрование канала передачи данных с S3.
* **Можно встроить BitTorrent протокол c** заменой http, как основного протокола скачивания файлов.
* **Предоставляется 99.999999999% гарантия целостности** и 99.99% гарантия доступности файлов в год (около 4-х часов недоступности в год).
* **S3 так же предполагает версионность файлов**.

Всегда можно восстановить файл предыдущей версии, т.е. откатиться до нужного состояния.

* **Пространство имен бакетов одно на всех пользователей**, поэтому названия бакетов должны быть уникальными.

**Amazon EBS**

EBS (Elastic Block Storage) — это один из типов

хранилища используемых в инстансах (виртуальных мащинах) EC2.

• 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.

**Возможности Amazon EBS**

* **Высокопроизводительные тома**. Можно выбрать тома на основе твердотельных (SSD) или жестких (HDD) дисков, чтобы обеспечить производительность, необходимую самым требовательным приложениям.
* **Доступность.** Каждый том Amazon EBS обеспечивает проектную доступность на уровне 99,999 % и автоматически реплицируется в своей зоне доступности, что гарантирует защиту приложений на случай отказа компонентов.
* **Шифрование. Шифрование** Amazon EBS эффективно поддерживает работу с данными при хранении и передаче между инстансами EC2 и томами EBS**.**
* **Управление доступом**. Гибкие политики управления доступом Amazon позволяют определять, кому разрешен доступ к томам EBS. Такой подход обеспечивает безопасный доступ к данным.
* **Снимки состояния.** Для защиты данных используются снимки состояния томов EBS на момент времени, которые направляются на долгосрочное хранение в Amazon S3.

Amazon Elastic File System

* Amazon Elastic File System (Amazon EFS) представляет собой простое масштабируемое файловое хранилище, предназначенное для использования с инстансами Amazon EC2 в облаке AWS.
* Amazon EFS **прост** в использовании благодаря удобному интерфейсу, с помощью которого можно быстро и легко создавать и настраивать файловые системы.
* При использовании Amazon EFS емкость хранилища **становится эластичной**, то есть увеличивается или уменьшается автоматически при добавлении и удалении файлов. В результате чего у приложений всегда будет необходимый объем хранилища.

Amazon Glacier

* [Amazon Glacier —](http://aws.amazon.com/glacier/) это безопасный, надежный и чрезвычайно экономичный **сервис хранения для архивации и долгосрочного хранения резервных копий данных**. Вы сможете безопасно хранить большие и малые объемы данных по цене всего **0,004 USD за гигабайт в месяц**, что намного выгоднее локальных решений.
* Чтобы **сохранить расходы на низком уровне** и удовлетворить различные потребности в извлечении данных, Amazon Glacier предлагает **три варианта доступа** к архивам, **от нескольких минут до нескольких часов**.

AWS Storage Gateway

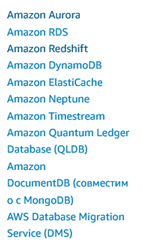
* Сервис [AWS Storage Gateway позв](http://aws.amazon.com/storagegateway/)оляет легко организовать **гибридное хранилище, используя локальные среды хранения и облако AWS**.
* Он объединяет многопротокольные устройства хранения данных и высокоэффективные сетевые подключения к сервисам облачных хранилищ Amazon и **обеспечивает уровень производительности, сопоставимый с локальным**, при практически неограниченном масштабе использования.
* Этот сервис можно **использовать в удаленных офисах и ЦОД** для рабочих нагрузок гибридного облака, включая миграцию данных, **быстрое масштабирование облака** и организацию многоуровневых хранилищ.

1. AWS. Сервис База данных, состав основных услуг. Сервис Миграция, состав основных услуг.

Сервисы Баз данных

**Базы данных AWS**

* **Специально разработанные базы данных для любых потребностей приложения**
* Реляционные б/д
* БД на основе пар «ключ-значение»
* Документные базы данных
* БД в памяти
* Графовые БД
* БД временных рядов
* Реестровые базы данных



Amazon Relational Database Service

**RDS** — это сервис баз данных, который выносится на отдельную машину.

* Проще говоря, это отдельные VPS серверы, оптимизированные для работы с базами данных.

Amazon Relational Database Service

В Amazon RDS доступны следующие СУБД:

* [**Amazon Aurora**](https://aws.amazon.com/rds/aurora/) **(**[MySQL](https://aws.amazon.com/rds/mysql/), [PostgreSQL](https://aws.amazon.com/rds/postgresql/))
* [**Amazon RDS**](https://aws.amazon.com/rds/) **(**[MySQL](https://aws.amazon.com/rds/mysql/), [PostgreSQL](https://aws.amazon.com/rds/postgresql/), [MariaDB](https://aws.amazon.com/rds/mariadb/), [Oracle](https://aws.amazon.com/rds/oracle/), [SQL Server](https://aws.amazon.com/rds/sqlserver/))
* [**Amazon Redshift**](https://aws.amazon.com/redshift/)

Благодаря технологиям машинного обучения, массово‑параллельному выполнению запросов и использованию столбчатых хранилищ на высокопроизводительных дисках производительность Redshift оказывается в **десять раз выше**, чем производительность других хранилищ данных.

Amazon Aurora

* **Amazon Aurora** — это **совместимое с MySQL и PostgreSQL** ядро реляционной базы данных, совмещающее в себе скорость и доступность дорогостоящих коммерческих баз данных с простотой и экономичностью баз данных с открытым кодом.
* **Производительность** Amazon Aurora **в пять раз выше**, чем производительность MySQL; она обеспечивает безопасность, доступность и надежность на уровне коммерческой базы данных, а стоит в десять раз меньше.
* **Преимущества** 
  + Высокая производительность. Amazon Aurora обеспечивает пропускную способность, превышающую пропускную способность стандартной базы данных **MySQL**, работающей на том же оборудовании**, в пять раз**, а стандартной базы данных **PostgreSQL** — в **два раза**. Вы получаете стабильно высокую производительность коммерческих баз данных по цене в десять раз ниже аналогов.
  + На самом крупном инстансе Amazon Aurora можно достичь производительности порядка **500 000 операций чтения и 100 000 операций записи в секунду**. Возможно дальнейшее масштабирование операций чтения с помощью реплик чтения с очень низкой задержкой, чуть более 10 миллисекунд.
  + Высокая масштабируемость. База данных Amazon Aurora масштабируется с инстанса с 2 виртуальными ЦПУ и 4 ГиБ памяти до инстанса с 32 виртуальными ЦПУ и 244 ГиБ памяти. Можно также добавить до 15 реплик чтения с минимальной задержкой в трех зонах доступности для дальнейшего масштабирования объема ресурсов чтения. По мере необходимости Amazon Aurora автоматически увеличивает объем хранилища с 10 ГБ до 64 ТБ.
  + · Высокая доступность и надежность. Amazon Aurora обеспечивает доступность выше 99,99 %. Восстановление при сбоях физического хранилища происходит незаметно для пользователя, а обработка отказа инстанса, как правило, занимает менее 30 секунд.

Amazon DynamoDB

* **Amazon DynamoDB** – это быстрая и гибкая служба баз данных NoSQL для всех приложений, требующих стабильной **задержки менее 10 секунд** при **работе в любом масштабе**. База данных является полностью управляемой и поддерживает как **документную** модель данных, так и модель «**ключ – значение**».
* Гибкая модель данных и надежная производительность позволяют использовать службу для **мобильных**, **игровых**, **высокотехнологичных** приложений, **интернет-приложени**й, «**Интернета вещей**» (IoT) и многого другого.
* ***Преимущества***
* **Быстрая и стабильная работа.** Решение Amazon DynamoDB работает стабильно и быстро в системах любого масштаба в любой области применения. Среднее **время обработки запроса на сервере составляет несколько миллисекунд**. По мере роста объемов данных и повышения необходимой производительности система Amazon DynamoDB обеспечивает соответствие требованиям к пропускной способности и времени обработки запроса с помощью технологий автоматического разбиения на разделы и SSD в системах любого масштаба.
* **Высокая масштабируемость.** При создании таблицы достаточно указать, какой объем ресурсов для обработки запросов вам требуется. Если необходимо изменить пропускную способность, просто обновите параметры таблицы с помощью консоли управления AWS или Amazon DynamoDB API. Amazon DynamoDB выполняет **все операции по масштабированию в скрытом режиме и в ходе их выполнения** продолжает обеспечивать соответствие установленным требованиям к пропускной способности.

Amazon ElastiCache

* Amazon ElastiCache – это веб-сервис, который **упрощает развертывание и масштабирование кэша в памяти в облаке**, а также управление им.
* Сервис **повышает производительность веб-приложений, позволяя получать информацию из быстрого**, управляемого кэша, размещенного в памяти, а не более медленных баз данных, размещенных на дисках.
* Amazon ElastiCache поддерживает **два модуля кэширования** в памяти с открытым исходным кодом:
  + **Redis** — отличающееся быстродействием хранилище данных в памяти с открытым исходным кодом и кэш.43 Amazon ElastiCache для Redis — это совместимый с Redis сервис в памяти, обеспечивающий удобство и мощь Redis с одной стороны и доступность, надежность и высокую производительность для самых требовательных приложений с другой.
  + **Memcached** – широко распространенная система кэширования объектов в памяти.45 Протоколы, используемые ElastiCache, совместимы с Memcached, поэтому все популярные инструменты, уже используемые в существующих средах Memcached, будут эффективно работать с этим сервисом.

БД на основе пар «ключ-значение»

* Базы данных «ключ – значение» оптимизированы для хранения и извлечения **пар ключей и значений** в больших объемах за миллисекунды, при этом они избавлены от присущих реляционным базам данных ограничений по масштабированию и чрезмерных затрат вычислительных ресурсов.
* **Сферы применения: приложения интернет-масштаба, ставки в режиме реального времени, корзины покупателей и настройки пользователей.**
* Предложение AWS - [**Amazon DynamoDB**](https://aws.amazon.com/dynamodb/)

Б/Д Документов

* Документные базы данных предназначены для хранения полуструктурированных данных, таких как документы, и интуитивно понятны разработчикам, поскольку данные обычно представлены в виде читабельных документов.
* **Сферы применения: управление контентом, персонализация и мобильные приложения.**
* Предложение AWS

[**Amazon DocumentDB (поддерживает совместимость с MongoDB)**](https://aws.amazon.com/documentdb/)

БД в памяти

* Для приложений, требующих доступ к данным в режиме реального времени, используются базы данных в памяти. Путем хранения данных прямо в памяти эти базы данных обеспечивают задержку в микросекундах, если миллисекунды – это слишком долго.
* **Сферы применения: кэширование, турнирные таблицы в играх и аналитика в режиме реального времени.**
* Предложения AWS:

[**Amazon ElastiCache для Redis**](https://aws.amazon.com/elasticache/redis/)

[**Amazon ElastiCache для Memcached**](https://aws.amazon.com/elasticache/memcached/)

Графовые БД

* Графовые базы данных используются для приложений, в которых миллионы пользователей могут создавать запросы и перемещаться между сложносвязными графовыми наборами данных с миллисекундной задержкой.
* **Сферы применения: обнаружение мошенничества, социальные сети и системы рекомендаций**
* Предложение AWS:

[**Amazon Neptune**](https://aws.amazon.com/neptune/)

БД временных рядов

* Базы данных временных рядов используются для эффективного сбора, синтеза и анализа огромных объемов данных, меняющихся со временем (их также называют данными временного ряда).
* **Сферы применения: приложения IoT, DevOps и промышленная телеметрия.**
* Предложение AWS:

[**Amazon Timestream**](https://aws.amazon.com/timestream/)

Реестр

* Реестровые базы данных используются, когда требуется централизованный доверенный узел, ведущий полный и подтверждаемый криптографическими средствами учет транзакций с возможностями масштабирования.
* **Сферы применения: системы записи, цепочки поставок, регистрация и банковские транзакции.**
* Предложение AWS:

[**Amazon Quantum Ledger Database (QLDB)**](https://aws.amazon.com/qldb/)

**SimpleDB**

* Amazon SimpleDB – это высокодоступное хранилище данных [NoSQL](https://aws.amazon.com/ru/nosql/), которое полностью обеспечивает администрирование баз данных.
* Разработчик с помощью запросов веб-сервисов сохраняет элементы данных и обращается к ним, а Amazon SimpleDB делает все остальное.
* **Типичные примеры использования** 
  + **Ведение логов**
  + **Интернет-игры**
  + **Индексация метаданных объектов Amazon S3**

**Функциональность SimpleDB**

* С помощью SimpleDB можно:[[1]](https://ru.wikipedia.org/wiki/Amazon_SimpleDB)
  + Создать новый домен для размещения уникального набора структурированных данных.
  + Выполнить операции GET, PUT и DELETE над элементами в вашем домене, с помощью пар атрибут-значение, которые связаны с каждым элементом. Сервис автоматически индексирует данные, которые добавляются в домен, чтобы их можно было быстро получить, нет необходимости заранее определять схемы или изменить их, если новые данные добавлены позже. Каждый элемент может иметь до 256 значений атрибутов. Каждый атрибут может варьироваться от 1 до 1024 байт.
  + Позволяет выполнять запросы с помощью SELECT [API](https://ru.wikipedia.org/wiki/API) или QUERY [API](https://ru.wikipedia.org/wiki/API) и с помощью набора операторов: =, !=, <, >, <=, >=, STARTS-WITH, AND, OR, NOT, INTERSECTION и UNION. Также есть возможность сортировки результатов с помощью оператора SORT. SimpleDB предназначен для использования приложений в реальном времени и оптимизировано для этого.
  + Платить только за потребляемые ресурсы.

**AWS сервисы миграции**

* AWS Application Discovery Service
* AWS Database Migration Service
* AWS Server Migration Service
* AWS Snowball
* AWS Snowball Edge
* AWS Snowmobile

AWS Application Discovery Service

* AWS Application Discovery Service помогает системным интеграторам быстро и **надежно планировать проекты миграции приложений**, автоматически идентифицируя приложения, которые выполняются в локальных ЦОД, связанные с ними зависимости и их профили производительности.

AWS Database Migration Service

* AWS Database Migration Service помогает легко и безопасно **переносить базы данных в AWS**. Исходная база данных во время миграции остается в полностью рабочем состоянии, что сводит к минимуму время простоя использующих ее приложений.
* AWS Database Migration Service позволяет **выполнить миграцию** данных, используя в качестве базы источника и целевой базы большинство распространенных баз данных, как коммерческих, так и с открытым исходным кодом (Oracle, MS SQL, MySQL, PostgreSQL, AWS Aurora**). Миграция выполняется** как **между однотипными**, так **и разными Б/Д.**

AWS Server Migration Service

* AWS Server Migration Service (SMS) — это безагентский сервис, упрощающий и ускоряющий **перенос тысяч локальных рабочих нагрузок в AWS**.
* AWS SMS позволяет автоматизировать операции, составлять расписание и отслеживать инкрементную **репликацию томов работающих серверов**, что облегчает координирование миграции крупных серверов.

AWS Snowball

* AWS Snowball — это **решение для переноса данных**, исчисляемых метаданными, предполагающее использование защищенных устройств для переноса крупных массивов данных в и из AWS.
* При использовании Snowball вам не придется писать код или приобретать дополнительное оборудование для передачи данных.
* Просто **создайте задание на консоли** управления AWS, и **устройство Snowball** будет предоставлено вам **автоматически**.
* По получении **подключите устройство к локальной сети**, загрузите и запустите клиент Snowball для установки подключения и с помощью клиента **выберите каталоги файлов**, которые требуется **передать на устройство**.
* Клиент **зашифрует файлы** и передаст их на устройство **на высокой скорости**.

AWS Snowball Edge

* AWS Snowball Edge — это **устройства переноса данных** объемом **100 ТБ** с —встроенными функциями хранения и вычисления.
* Его можно использовать для обмена большими объемами данных с AWS, в качестве временного хранилища или для поддержки удаленных или отключенных от Интернета объектов.
* Подключение Snowball Edge к существующим приложениям и инфраструктуре выполняется с помощью стандартного интерфейса хранилища.
* Это облегчает весь процесс переноса данных, поскольку настройка и интеграция не занимают много времени. Объединив **несколько устройств Snowball Edge**, можно создать **уровень локального** хранения для локальной **обработки данных**. Это обеспечит приложению возможность продолжать работу даже при отсутствии доступа в облако.

AWS Snowmobile

* AWS Snowmobile — это сервис для переноса данных, исчисляемых **эгсабайтами**, позволяющий переносить чрезвычайно крупные объемы данных в AWS.
* Для перемещения данных используется транспортный **контейнер Snowmobile длиной 13,7 м** со специальной защитой, перевозимый при помощи грузового автомобиля с прицепом.
* В таком контейнере можно перемещать **до 100 ПБ** данных. С помощью этого сервиса можно легко перемещать огромные объемы данных, такие как видеобиблиотеки и репозитории изображений, а также выполнять миграцию всего центра обработки данных.
* Перенос данных с помощью Snowmobile отличается **высокой безопасностью, скоростью и экономичностью**.

1. Хранение данных в AWS. Apache Hadoop в Amazon EMR.

**Apache Hadoop в Amazon EMR**

* **Apache™ Hadoop**® – это программный проект с открытым исходным кодом, предназначенный для эффективной обработки больших наборов данных
* **Файловая система EMR File System (EMRFS**), которая применяется в кластерах Amazon EMR, позволяет использовать сервис Amazon S3 в качестве уровня хранения для Hadoop.
* **Среду Hadoop** можно без труда интегрировать с другими сервисами, например с [Amazon S3](https://aws.amazon.com/ru/s3/), [Amazon Kinesis](https://aws.amazon.com/ru/kinesis/), [Amazon Redshift](https://aws.amazon.com/ru/redshift/) и [Amazon DynamoDB](https://aws.amazon.com/ru/dynamodb/), для перемещения данных, создания рабочих процессов и выполнения аналитики с использованием различных сервисов платформы AWS.
* **Дополнительно можно использовать каталог данных** [**AWS Glue**](https://aws.amazon.com/ru/glue/)в качестве управляемого репозитория метаданных для Apache Hive и Apache Spark.

**Гибкие возможности хранения данных**

Amazon EMR предоставляет управляемую инфраструктуру Hadoop, которая способна эффективно, быстро и экономично обрабатывать большие объемы данных на динамически масштабируемых инстансах Amazon EC2. В EMR можно также запускать другие известные распределенные инфраструктуры, включая [Apache Spark](https://aws.amazon.com/emr/details/spark/), [HBase](https://aws.amazon.com/emr/details/hbase/), [Presto](https://aws.amazon.com/emr/details/presto/) и [Flink](https://aws.amazon.com/blogs/big-data/use-apache-flink-on-amazon-emr/)

* С Amazon EMR можно использовать разные хранилища данных, в том числе Amazon S3, распределенную файловую систему Hadoop (HDFS) и Amazon DynamoDB.

1. Платформа Microsoft Microsoft Azure. Обзор служб Microsoft Azure.

**Microsoft Azure**

* Microsoft Azure — это открытая и гибкая облачная платформа, позволяющая быстро создавать, развертывать и управлять приложениями в глобальной сети центров обработки данных под управлением Майкрософт.
* Приложения можно разрабатывать с помощью любого языка, инструмента или платформы.
* Кроме того, приложения в общедоступном облаке можно интегрировать с имеющейся ИТ-средой предприятия.

Платформа Azure Resource Manager

* Это - **техническая платформа** и **сервис оркестрации**, связывающий все компоненты Azure. Она объединяет поставщиков ресурсов, ресурсы и **группы ресурсов Azure**, образуя тесно связанную облачную платформу.
* ARM :
  + делает сервисы Azure доступными для подписок,
  + определенные типы ресурсов — для групп ресурсов,
  + обеспечивает порталу и другим клиентам возможность доступа к ресурсам и интерфейсам API ресурсов
  + выполняет аутентификацию при осуществлении доступа к ресурсам.
  + позволяет пользоваться такими функциями, как маркировка, аутентификация, **управление доступом на основе ролей** (**RBAC**), блокировка ресурсов и обеспечение применения политик к подпискам и их группам ресурсов.
  + Кроме того, она предоставляет функции развертывания и управления с использованием портала Azure, Azure PowerShell и средств **командной строки** (**CLI**)

**Службы Azure по категориям (2020)**

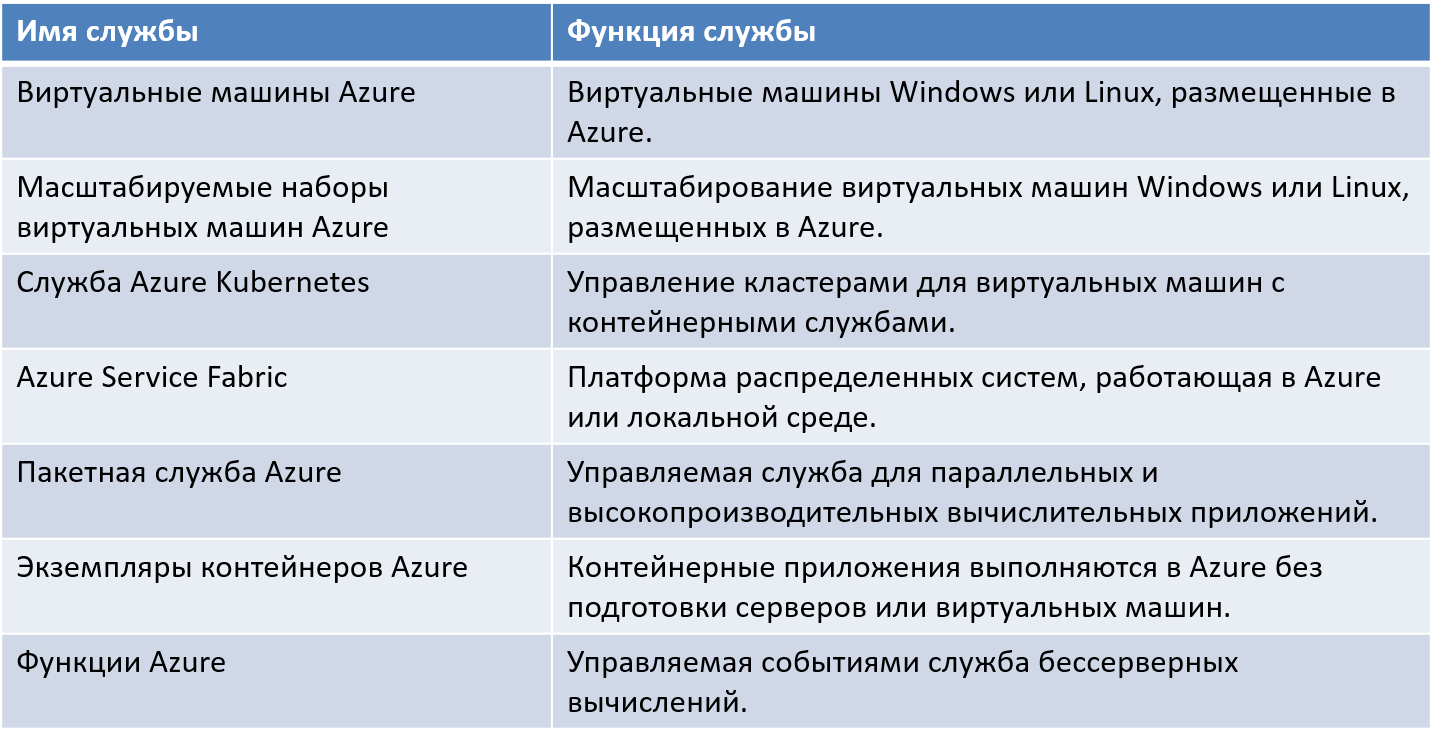
* Общие (15 служб)
* Вычисления (23)
* Сети (27)
* Хранилище (11)
* Мобильный (2)
* Контейнеры (7)
* Базы данных (16)
* Аналитика (14)
* Искусственный интеллект и машинное обучение (8)
* "Интернет вещей" (23)
* Интеграция (21)
* Удостоверение (14)
* Безопасность (7)
* DevOps (6)
* Перенос (6)
* Управление + система управления (23)
* Intune (17)
* Другое (32)

***Наиболее популярные***:

Вычисления, Сеть, Хранилище, Мобильные приложения, Базы данных, Веб-службы, Интернет вещей, Большие данные, ИИ, DevOps

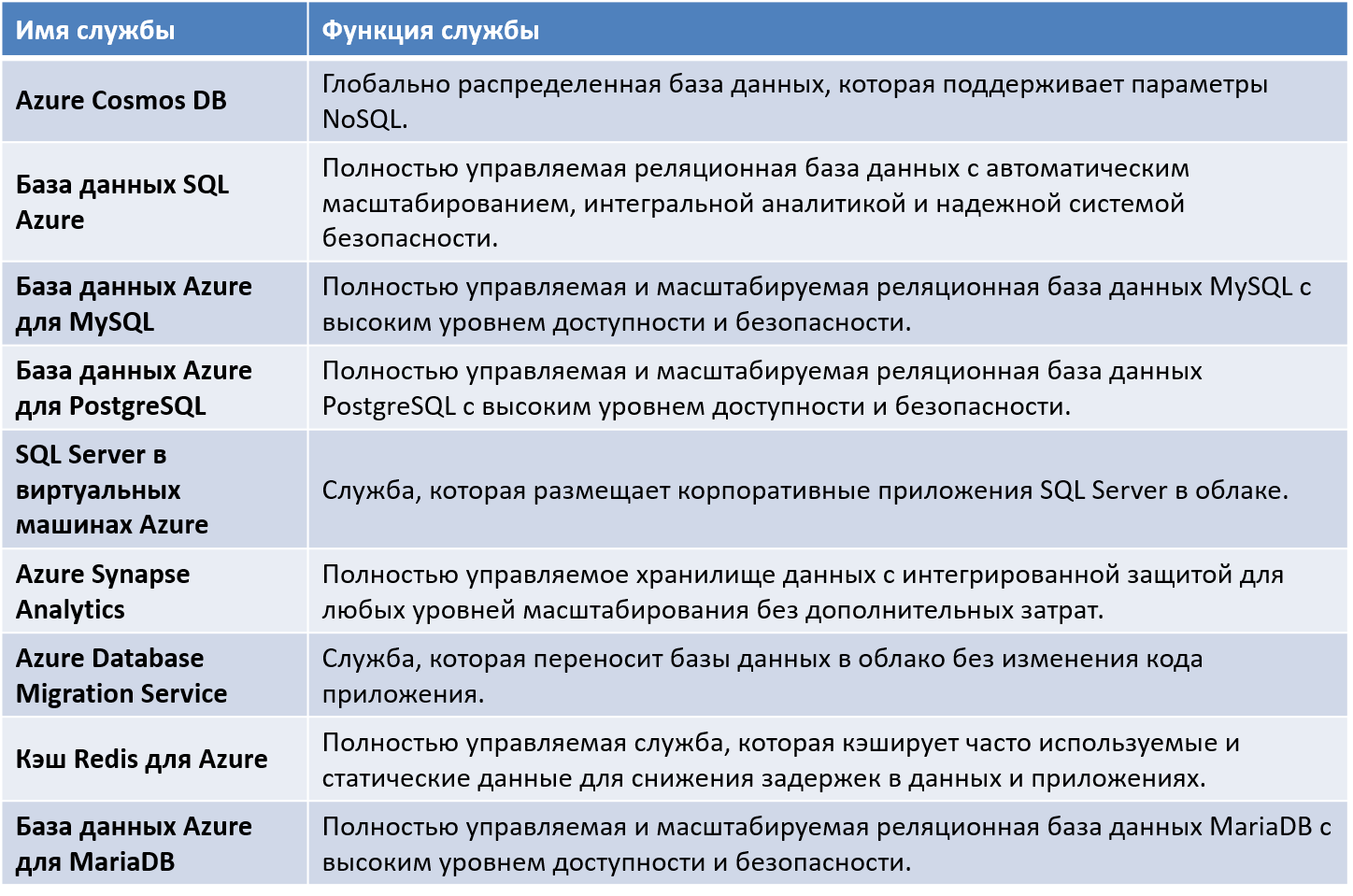
1. Вычислительные службы Microsoft Azure. Службы базы данных Microsoft Azure.

**Службы вычислений**



Службы вычислений — это одна из основных причин того, что компании переходят на платформу Azure.

Azure предоставляет широкий выбор вариантов для размещения приложений и служб.

**Базы данных**

1. Обеспечения высокой доступности в Azure. Набор доступности. Домены отказа и обновлений. Зона доступности.

**Одна точка отказа**

* Приложение который работает на одной виртуальной машине не может обеспечить бесперебойную работу так оно имеет единую точку отказа. Если эта одна виртуальная машина станет недоступно, недоступным станет и приложение работающее на этой машине.

**Методы обеспечения высокой доступности в Azure**

* Все центры обработки данных Azure созданы для обеспечения высокой доступности. Резервные источники питания, несколько сетевых подключений и запасные массивы ивы хранения данных - это лишь некоторые из основных концепции избыточности, которые предоставляет Azure и управляет ими.
* Однако вся это избыточность ресурсов предоставляемая приложениям не может помочь, если приложение исполняется на одной виртуальной машине.
* Чтобы обеспечить высокую доступность приложения в Azure реализованы следующие методы гарантирующие такую доступность для рабочих нагрузок IaaS:
  + Группы доступности;
  + Домен сбоя;
  + Домен обновления;
  + Зона доступности.

**Резервирование ВМ с помощью группы доступности**

* **Группа доступности** (***availability set***) – это метод логического группирования виртуальных машин работающих в одной зоне, путем распределения их по одной в разных стойках центра обработки данных Azure.
* Azure предоставляет ресурс под названием «группа доступности», и связь нескольких виртуальных машин с этим ресурсом обеспечивает им высокую доступность, так как они размещаются на отдельных физических стойках в центре обработки данных Azure.
* Для достижения этой цели группы доступности содержат **домен сбоя** и **домен обновления**.
* Важно отметить, что группы доступности реализуют высокую доступность в пределах центра обработки данных

**Домены сбоев (отказов)**

* **Домен сбоя** (**FD**) — это группа ВМ, которые используют общий источник питания и сетевой коммутатор.
* Домены сбоя обеспечивают высокую доступность в случае внепланового простоя из-за сбоя оборудования, питания или сети. Пользователи не могут настраивать домены сбоя.
* Когда виртуальная машина создается и включается в группу доступности, она размещается в домене сбоя.
* По умолчанию каждая группа доступности имеет 2 или 3 домена сбоя, в зависимости от регионов Azure.
* Домены сбоя связаны с **физическими стойками** в центре обработки данных Azure.

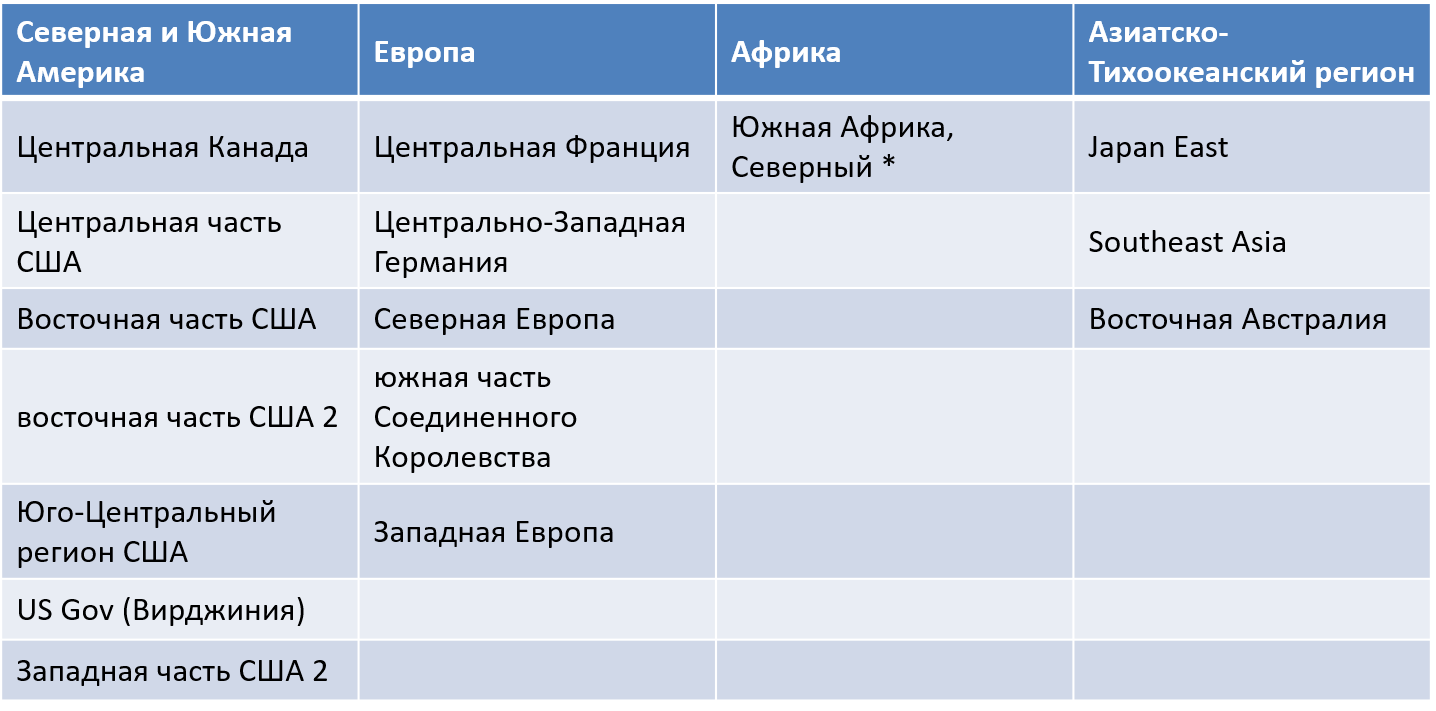
**Домены отказа**

* Домен отказов - это логическая группа оборудования в центре обработки данных Azure.
* Формирование домена отказов – это способ резервирование аппаратных средств на случай различных видов аварийных ситуаций, которые могут возникнуть в процессе работы ЦОД:
  + Выход из строя аппаратных средств серверов;
  + Отказы в системах электропитания и охлаждения;
  + Отказы в работе сетевого оборудования;
  + Отказы в работе систем хранения на которых располагаются управляемые диски ВМ.
* Домен отказов содержит оборудование с общим питанием или сетевым оборудованием.
* Платформа Azure логически назначает кластеры хранения доменам сбоя, чтобы гарантировать, что управляемые диски виртуальных машин, распределенных по доменам сбоев, также распределяются по соответствующим кластерам хранения.
* Azure сам формирует эти домены и отслеживает процесс распределения вновь создаваемых виртуальных машин по доменам отказа.
* **Пользователи не контролируют** наличие и формирование доменов отказа. **Azure обеспечивает**, чтобы для приложений пользователя всегда были доступные виртуальные машины, даже если откажет электропитание или сеть в отдельных стойках ЦОД.

**Домены обновлений**

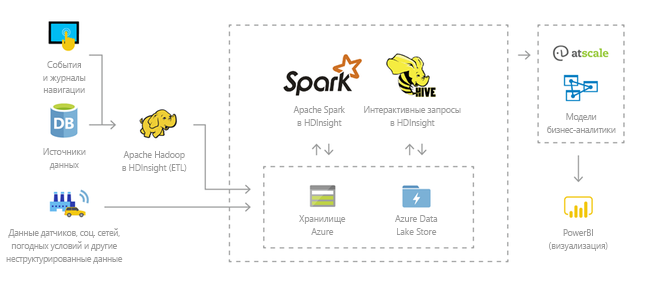
* Помимо аппаратных отказов, причиной неработоспособности ВМ может быть проведение операций по обновлению ПО, установке патчей ОС или проведение иных работ связанных с проведением планового обслуживания программных и аппаратных средств ЦОД.
* Для **защиты от простоев** в этих случаях в Azure формируются соответствующие логические группы - **домены обновлений**.
* Инженеры Azure выполняют (в основном автоматизированное) обслуживание и применяют обновления на всем физическом оборудовании только в одном домене обновления, а затем проводят такие же операции обслуживания всего оборудования в следующем домене обновления.
* Такой порядок проведения обслуживания применяется для того, **чтобы не все виртуальные машины**, входящие в одну **группу доступности** были **остановлены одновременно** при выполнении **технического обслуживания**.

**Регионы Azure поддерживаающие зоны доступности**



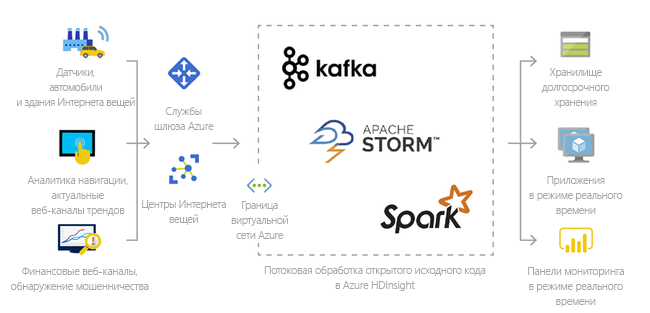
1. Работа с большими данными в Azure. Azure HDInsight и стек технологий Apache Hadoop. HDInsight и Интернет вещей.

**Azure HDInsight и стек технологий Apache Hadoop**

****

* При помощи HDInsight вы можете выполнять интерактивные запросы структурированных и неструктурированных данных в любом формате и объемом в несколько петабайт.
* Также можно создавать модели и подключать их к средствам бизнес-аналитики.
* Azure HDInsight можно применять в различных сценариях обработки больших данных. Это могут быть исторические данные (т. е. собранные и сохраненные) или данные в реальном времени (потоковые данные, передаваемые прямо из источника).

HDInsight и **Интернет вещей**

****

* HDInsight можно использовать для обработки потоковых данных, получаемых в режиме реального времени с различных устройств.

1. Система виртуализации с открытым кодом OpenNebula.

### ****OpenNebula: система виртуализации с открытым кодом****

**OpenNebula** — это открытая платформа управления облачными и виртуализированными инфраструктурами, предназначенная для создания частных, публичных и гибридных облаков. Она предоставляет инструменты для управления виртуальными машинами, контейнерами, хранилищами данных и сетевыми ресурсами в централизованной системе.

### ****1. Основные возможности OpenNebula****

1. **Управление виртуализацией**  
   Поддержка гипервизоров, таких как KVM, VMware, LXC. Позволяет централизованно управлять виртуальными машинами и их жизненным циклом.
2. **Поддержка гибридного облака**  
   Интеграция с публичными облаками (например, AWS, Microsoft Azure, Google Cloud) для масштабирования ресурсов.
3. **Контейнеризация**  
   Поддержка Docker и LXC для работы с контейнерами в облачной среде.
4. **Управление сетями**  
   Включает возможности для создания виртуальных сетей, балансировки нагрузки, настройки VPN и фаерволов.
5. **Управление хранилищем**  
   Интеграция с локальными и удалёнными хранилищами, такими как NFS, Ceph, GlusterFS, iSCSI.
6. **API и автоматизация**  
   Поддержка RESTful API для автоматизации задач и интеграции с DevOps-инструментами.
7. **Масштабируемость и отказоустойчивость**  
   Возможность масштабирования для поддержки больших распределённых инфраструктур.
8. **Графический интерфейс (Sunstone)**  
   Удобная панель управления для администраторов и пользователей.

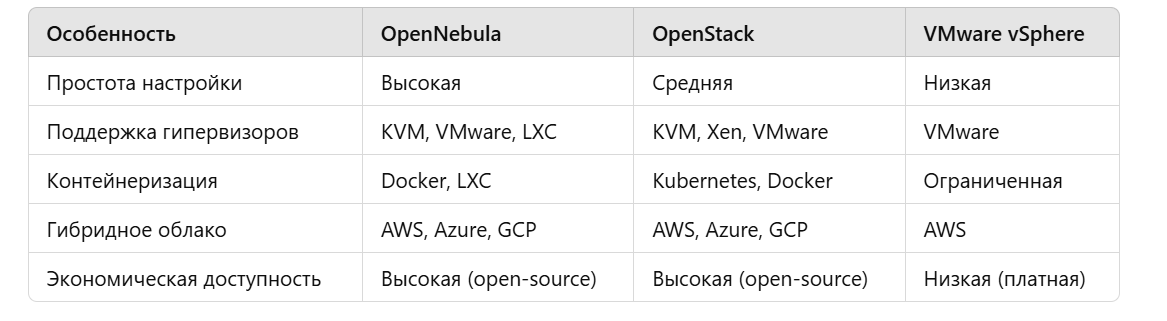
### ****2. Архитектура OpenNebula****

1. **Frontend**  
   Центральный компонент, управляющий всей системой. Отвечает за планирование, мониторинг, управление виртуальными машинами, сетями и хранилищами.
2. **Nodes**  
   Узлы, на которых выполняются виртуальные машины и контейнеры. Они управляются через гипервизоры или контейнерные технологии.
3. **Datastores**  
   Места хранения образов виртуальных машин, данных и шаблонов.
4. **Virtual Networks**  
   Виртуальные сети, обеспечивающие подключение виртуальных машин и управление сетевыми ресурсами.
5. **Интерфейсы взаимодействия**
   * **CLI (Command-Line Interface)** для администраторов.
   * **Sunstone** — веб-интерфейс для управления облаком.
   * **API** для интеграции с внешними системами.

### ****3. Преимущества OpenNebula****

1. **Открытый исходный код**  
   Возможность изучения и модификации системы под конкретные нужды.
2. **Простота установки и настройки**  
   По сравнению с конкурентами (например, OpenStack), OpenNebula имеет более простой процесс развёртывания.
3. **Универсальность**  
   Поддержка как гипервизоров, так и контейнеров, что делает её подходящей для разных типов нагрузок.
4. **Лёгкость интеграции**  
   Возможность работы с различными публичными облаками и системами хранения.
5. **Экономическая эффективность**  
   Отсутствие высоких затрат на лицензии, особенно для малого и среднего бизнеса.

### ****4. Сравнение с другими платформами****



### ****5. Основные случаи использования OpenNebula****

1. **Частные облака**
   * Создание инфраструктуры для внутреннего использования в компании.
   * Управление виртуальными машинами и контейнерами.
2. **Гибридные облака**
   * Расширение локальных ресурсов за счёт интеграции с публичными облаками.
3. **Публичные облака**
   * Обеспечение многопользовательского доступа к вычислительным ресурсам.
4. **Тестовые среды**
   * Развёртывание изолированных окружений для тестирования и разработки.

### ****6. Проблемы и вызовы****

1. **Ограниченная функциональность**  
   В сравнении с более крупными платформами, такими как OpenStack, OpenNebula может не покрывать потребности крупных корпораций.
2. **Зависимость от сообщества**  
   Основная поддержка идёт от сообщества, что может быть критичным для компаний, нуждающихся в гарантированной техподдержке.
3. **Масштабируемость**  
   Для очень крупных облачных инфраструктур OpenNebula может уступать конкурентам.

### ****7. Примеры использования****

1. **Телекоммуникационные компании**  
   OpenNebula используется для управления частными облаками, обработки больших данных и автоматизации процессов.
2. **Образовательные учреждения**  
   Создание виртуальных лабораторий для студентов.
3. **Малый и средний бизнес**  
   Экономичное решение для управления IT-инфраструктурой.
4. **Исследовательские центры**  
   Обеспечение гибкости и масштабируемости для вычислительных задач.

### ****8. Заключение****

**OpenNebula** — это мощная, простая в использовании и экономически эффективная платформа виртуализации и управления облаками. Она подходит для компаний, которые ищут универсальное и гибкое решение для создания частных, публичных или гибридных облаков, с минимальными затратами на внедрение и обслуживание.

1. Причины появления культуры DevOps. Цели и задачи DevOps.

DevOps - это новый термин

Суть DevOps - это **культура разработки программного обеспечения**, объединяющая программное обеспечение разработки (dev) и операции с программным обеспечением (ops).

Есть много разных интерпретаций того, что это объединение действительно означает.

Означает ли это, что каждый должен быть разработчиком?

Означает ли это разработчики в конечном итоге несут ответственность за непрерывную работу конечных продуктов?

Чтобы понять эти вопросы, вы должны сначала понять, почему существует DevOps

**Цели и задачи DevOps**

* Поскольку процессы девопс охватывают весь цикл поставки ПО, выделяют несколько **основных целей этого подхода**:
  + сокращение времени для выхода на рынок;
  + снижение частоты отказов новых релизов;
  + сокращение времени выполнения исправлений;
  + уменьшение количества времени на восстановления при сбое новой версии или других случаях отключения текущей системы.

**Причины появления Devops**

**Методология AGILE**

* Agile (agile software development, от англ. agile – проворный) – это семейство «гибких» подходов к разработке программного обеспечения. Такие подходы также иногда называют фреймворками или agile-методологиями.
* Все современные методологии проектирования ПО включая Agile используют **спиральную модель** ЖЦ проекта.
* К отдельным agile-подходам относятся scrum и kanban.
  + **Scrum – это «подход структуры»** рабочий процесс делится на равные спринты – обычно это периоды от недели до месяца, в зависимости от проекта и команды. Перед спринтом формулируются задачи на данный спринт, в конце – обсуждаются  результаты, а команда начинает новый спринт. Спринты очень удобно сравнивать между собой, что позволяет управлять эффективностью работы.
  + **Kanban – это «подход баланса».** Его задача – сбалансировать разных специалистов внутри команды и избежать ситуации, когда дизайнеры работают сутками, а разработчики жалуются на отсутствие новых задач.

Вся команда едина – в kanban нет ролей владельца продукта и scrum-мастера. Бизнес-процесс делится не на универсальные спринты, а **на стадии выполнения конкретных задач**: «Планируется», «Разрабатывается», «Тестируется», «Завершено» и др.

**Жизненный цикл проекта:  
Этапы тестирование и ввод в действие**

* Разработка проходит через ряд циклов — итераций. Каждая итерация — это фактически отдельный проект, где разрабатывают фрагмент программы, улучшают функциональность, добавляют новые возможности.
* Этап тестирования предполагает выполнение работ (разработчик ПО):
  + Создание инфраструктуры тестирования и ее конфигурирование
  + Развертывание и конфигурирование тестируемого ПО
  + Выполнение тестирования (проведение испытаний).
* Этап ввод в действие предполагает выполнение работ (сисадмин):
  + Создание инфраструктуры производственной среды ИС заказчика и ее конфигурирование
  + Развертывание и конфигурирование созданного ПО
  + Проверка работоспособности на предмет полноты и качества реализации требований ТЗ.

**Творчество vs. порядок**

* **Разработка ПО** — творческий процесс. Управление творческим процессом предполагает гибкость и умение не мешать.
* Собственно, методология Agile предназначена именно из этого:
  + ее цель — управлять хаосом, минимально его упорядочивая. И это, действительно хорошо: при правильно примененном подходе Agile получается неплохой продукт.
* **Эксплуатация** же, напротив, требует абсолютно четкого знания обо всех элементах системы, конфигурациях, потенциальных точках нестабильности и т. п. Здесь властвует ITIL и ему подобные методологии.
* Конфликт подходов к настройке среды развертывания:
  + Разработчик: - «нам проще написать свое, чем разбираться в чужом»
    - Разработчику проще написать 100 строк кода, чем разбираться в документации и конфигах.
    - Разработчик не будет разбираться в конфигурации инфраструктуры и сделает все «по умолчанию» + по советам со StackOverflow.
  + Сисадмин: - создание требуемой конфигурации окружения из нескольких разных систем выполняется только с помощью:
    - Скриптов и Параметров окружения, или с помощью того и другого.
    - Админ не полезет в код, пока не поймет, что это единственно возможный вариант, точно так же как разработчик не полезет в конфиги.
* Налицо вывод:
  + Сисадмин и программист разработчик **имеют абсолютно разные компетенции**.

**Разработка против эксплуатации**

* Традиционно разработка и эксплуатация определяют два разных набора навыков, требующих разных профессионалов.
* Два набора профессионалов часто образуют конфликтующие “племена” внутри компании, потому что у них конфликтующие приоритеты.
* Это вызывает множество ненужных трений.

**Изменения против стабильности**

* Программным инженерам необходимо постоянно вносить изменения, чтобы включать новые функции, исправлять ошибки, оптимизировать производительность разрабатываемого ПО.
* Администраторам (ИТ-специалистам) необходимо поддерживать стабильность среды для непрерывной работы.
* По мнению разработчиков программного обеспечения, невозможность выпускать обновления без быстрой адаптации инфраструктуры, что замедляет их работу.
* Администраторы считают, что частые изменения инфраструктуры нарушают их работу.
* Этот конфликт приоритетов кажется **непримиримым** и вызывает постоянные трения между двумя командами.

**Среда разработки и производственная среда**

* Многие конфликты между разработкой и эксплуатацией вызваны несоответствиями между разными средами.
* Программное обеспечение часто отлично работает в среде разработки и ломается когда оно запускается в производственном окружении.
* Помимо проблем с масштабируемостью, многие проблемы производственной среды вызваны различиями в конфигурации, такими как разные версии ПО, зависимости версиий, разные уровни исправлений ОС, разные аппаратные стеки или просто разные конфигурации.

**Контроль качества (QA)**

* В гражданской войне между разработкой и эксплуатацией **жертвой** оказывается обеспечение качества (QA).
* Специалистов группы контроля качества (QA) часто обвиняют в том, что они **не нашли очевидных ошибок, которые просочились в производственную среду**.
* Их проблемы часто сводятся к тому, что среда контроля качества, которую они используют, часто отличается от среды разработки и производственной среды.
* Это еще больше усложняет ситуацию:
  + когда производственная среда не может быть воспроизведена ни в среде разработки, ни в тестовой среде контроля качества.

**Отладчики против журналов**

* Диагностика проблем продукта часто более сложна, чем отладка в среде разработки.
* Разработчик может использовать богатый набор инструментов отладки в среде разработки.
* Однако часто требуется ручное чтение журналов в качестве единственного метода устранения неполадок.
* В мультитенантной распределенной системе, поиск и сбор необходимых журналов - непростая задача.
* Администраторы, часто недовольны когда разработчики хотят подключиться к производственным серверам, чтобы покопаться в системных журналах.
* В свою очередь разработчики также недовольны, когда им приходится преодолевать препятствия, чтобы получить доступ к файлам журналов с производственных серверов.

**Детализация журналов**

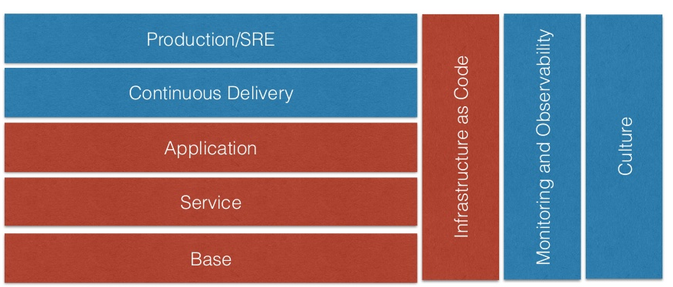
* Разработчикам часто нужны журналы с высоким уровнем детализации, чтобы фиксировать сведения об ошибках.
* Сисадмины часто предпочитают менее детализированные журналы, чтобы снизить потребление ресурсов на производственных серверах.
* Так что даже когда журналы есть, разработчики часто не могут собрать информацию из журналов производства.
* А так как, конфиденциальность данных становится все более серьезной проблемой, компания часто усиливает ограничительную политику при доступе к журналам.

1. DevOps. Инфраструктура как код. Изоляция вариантов развертывания.

**Инфраструктура как код**

* Идея IaC (Infrastructure as Code) состоит в том, чтобы явно зафиксировать все требования к инфраструктуре в виде недвусмысленного кода на некотором формальном языке:
  + YAML, JSON, HCL
  + Язык DSL (Chef, Puppet, Ansible и т.д.)
  + Ruby, Python, Go, PowerShell, …
* Код описания конфигурации не подлежит интерпретации человеком и может точно выполняться компьютерами, в результате обеспечивается согласованность конфигураций разных сред развертывания ПО.
* Сегодня IaC это основа облачных вычислений и неотъемлемая часть DevOps.

**Инфраструктурные слои**



* Инфраструктурный репозиторий включает:
  + Репозиторий с инфраструктурным кодом
  + Иерархия и слои – окружение, роли, модули
  + Внешние зависимости
  + Переменные
  + Инструменты и конфигурации для инструментов
  + Тесты и документация

**Изоляция вариантов развертывания ПО**

* Избежать конфликтов между разными версиями разворачиваемого ПО, а также проблем с совместимостью ПО узлов с развертываемым ПО можно используя возможности изоляции разных версий развертываемого ПО с помощью применения технологий виртуализации:
  + Виртуальных машин
  + Контейнеров
* Виртуальные машины:
  + идеальное решение, позволяющее быстро разворачивать требуемую среду развертывания, быстро вносить изменения, легко устранять отказы и аварии ВМ, путем ее перезапуска из образа.
  + Недостатки ВМ:
    - Образы это двоичные файлы ВМ поэтому их трудно использовать в системах управления версиями.
    - Из-за больших размеров ВМ не удается обеспечить высокую плотность размещения приложений на одном физическом сервере.

**изоляция ПО с помощью Контейнеров**

* Контейнеры:
  + обеспечивают легкую (в смысле потребляемых ресурсов) изоляцию, которая позволяет ядру системы повторное использование модулей и драйверов ОС в пакетах приложений, которые изолированы такими методами, как cgroups, пространства имен (namespaces) и chroots.
  + запуск контейнера - это просто запуск процесса, а не загрузка ОС и приложений, что позволяет быстро запускать и уничтожать контейнеры.
  + поскольку текстовые файлы могут описывать контейнеры, для контейнеров можно применять ПО контроля версий, а также автоматизированные конвейеры сборки для описаний контейнеров
  + Недостатки конвейеров:
    - Изоляция контейнеров слабее, чем у виртуальных машин, но во многих сценариях хостинга этого достаточно, когда доверенное программное обеспечение использует один и тот же хостинг-ресурс.

1. Концепция DevOps CI/CD. Методология непрерывного развертывания. Инструменты DevOps. Конвейер DevOps.

**методология непрерывного развертывания**

* Методология непрерывного развертывания (Continuous Delivery) позволяет быстрее и надежнее, передавать программное обеспечение в эксплуатацию.
* Манифест гибкой разработки программного обеспечения определяет следующий принцип как наиболее важный:

*«Наивысшим приоритетом для нас является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения».*

**Continuous Integration & Continuous Delivery – конвейер для создания качественного ПО**

* **Непрерывная интеграция** (Continuous Integration, CI) и **непрерывное развертывание** (Continuous Delivery, CD) — это концепция методологии DevOps.
* Цель - уменьшить количество ошибок в программном коде, ускорить процесс их выявления и повысить качество выпускаемого продукта.
* Cуть  CI\CD в регулярном изменении программного кода (по несколько раз в день).
* Далее программный код вносится в репозиторий с последующей сборкой и его тестирования.
* Тестирования проводятся после каждого изменения программного кода.
* Данная концепция позволяет обнаружить ошибки на более раннем этапе, что поможет сэкономить время на его устранение, ведь чем позднее найдена ошибка, тем труднее ее исправить.
* И, конечно, без доставки ПО до потребителя сложно себе представить использование ПО. Поэтому CD является обязательной частью для хорошо работающего DevOps.

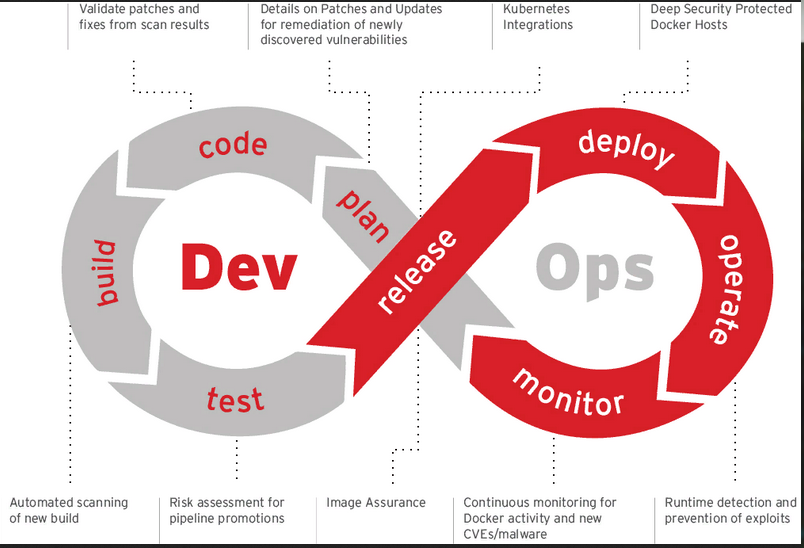
**Непрерывное развертывание Предполагает:**

* Развертывание осуществляется чаще — вплоть до нескольких раз в день. Это сокращает время ввода в действие новых возможностей.
* Частые развертывания ускоряют получение отзывов на новые особенности и изменения в коде. Разработчикам не приходится вспоминать, что делалось в прошлом месяце.
* Чтобы развертывание протекало быстрее, **создание тестового окружения** и собственно тестирование должны осуществляться **автоматически**, иначе на это будет уходить слишком много сил.
* Автоматизация улучшает воспроизводимость окружения: если тестовое окружение было благополучно создано, ту же автоматизированную процедуру можно использовать для создания рабочего окружения практически с той же конфигурацией. Как следствие, проблемы, вызванные ошибками в конфигурации, не будут возникать в рабочем окружении.
* Кроме того, автоматизация дает больше гибкости. Тестовые окружения можно создавать по мере необходимости.
* Автоматизация тестирования упрощает воспроизведение ошибок. Поскольку во время каждого теста выполняется одна и та же последовательность действий, с процедурой выполнения теста не связано никаких ошибок.

**Какие есть инструменты у DevOps**

* Каждый инструмент решает определенные задачи. Вот примеры некоторых из них:
  + Облачная инфраструктура – Amazon Web Services (AWS), Microsoft Azure
  + Система контроля версий – GitHub
  + Непрерывная интеграция – TeamCity, Jenkins, Gitlab CI
  + Инструмент для мониторинга – Prometheus, Zabbix, Nagios
  + Инструменты для работы с контейнерами – CoreOS, Docker, Kubernetes

**Конвейер Devops**



1. Реализация DevOps в AWS.

**AWS:  
Службы и Инструменты Devops**

**Службы DevOps на AWS**

* **AWS CodeCommit** — это полностью управляемая служба контроля исходного кода, реализация GIT, полностью его повторяющая. Сам код храниться на S3.
* **AWS CodeBuild** - сборочный сервер — для проектов, требующих **сборки** перед развёртыванием, например, Java. По умолчанию запускает контейнер на базе Ubuntu, но можно указать и свой собственный. Может быть использован в качестве тестового.
* **AWS CodeDeploy - с**ервис для развёртывания кода, который с помощью предустановленного агента и гибких настроек работает в любом окружении. Особенным отличием является то, что агент работает не только с Amazon AWS виртуалками, но и «внешними», что позволяет централизованно разворачивать самое разношерстное ПО, в т.ч. и локально.
* **AWS CodePipeline –** конвейер развертывания, взаимодействует с предыдущими тремя сервисами, запуская их в нужной последовательности и, собственно, обеспечивая автоматизацию DevOps-процессов.
* **AWS CodeStar** - сервис, по сути дублирующий **CodePipeline**, но ориентрированный на простоту запуска и настройки, что решается с помощью широкого **набора готовых шаблонов** (под связку приложение-язык) и действительно удобного Dashboard с плагинами, имеющими некоторую интеграцию с сервисом мониторинга (**CloudWatch**) и плагином для интеграции с Jira.

**AWS CodeCommit**

* AWS CodeCommit — это полностью управляемая служба контроля исходного кода, упрощающая размещение защищенных и в высшей степени масштабируемых частных репозиториев Git.
* AWS CodeCommit устраняет необходимость в использовании собственной системы контроля исходного кода.

**AWS Codecommit**

* AWS CodeCommit – это полностью управляемый сервис по управлению исходным кодом, предназначенный для хостинга защищенных репозиториев на основе Git.
* Он упрощает совместную работу команд с кодом в безопасной системе с высокой масштабируемостью. С сервисом CodeCommit не требуется поддерживать собственную систему управления исходным кодом или беспокоиться о масштабировании соответствующей инфраструктуры.
* С помощью CodeCommit можно безопасно хранить любые ресурсы, от исходного кода до исполняемых файлов.
* Сервис эффективно интегрируется с существующими инструментами Git.

**AWS CodeBuild**

* AWS CodeBuild — это полностью управляемый сервис сборок, который **компилирует** исходный код, **запускает тесты** и создает готовые к развертыванию **пакеты** программного обеспечения.
* Благодаря CodeBuild больше не нужно распределять и масштабировать собственные серверы сборок, а также обеспечивать управление ими.
* CodeBuild непрерывно масштабируется и способен обрабатывать **несколько сборок одновременно**, поэтому сборки не будут ждать в очереди.

**AWS CodeBuild  
Manager**

* AWS CodeBuild – это полностью управляемый сервис:
* создания сборок,
* выполняющий компиляцию исходного кода,
* проводящий тестирование и
* формирующий готовые к развертыванию пакеты программного обеспечения.

**AWS CodeDeploy**

* AWS CodeDeploy — это сервис **автоматизации развертываний** кода в любом инстансе, включая инстансы EC2 и инстансы, выполняемые локально.
* AWS CodeDeploy упрощает быстрый выпуск новых возможностей, помогает избежать простоев во время развертывания и **упрощает весь процесс обновления** приложений.
* AWS CodeDeploy можно использовать для автоматизации развертываний ПО, избавляясь от необходимости выполнять операции вручную с риском совершения ошибки.
* Этот сервис **масштабируется** вместе с вашей инфраструктурой, так что вы легко сможете выполнить развертывание в одном или **тысячах** инстансов.

**AWS CodeDeploy**

* Cервис для **автоматизации развертывания программного обеспечения** с использованием различных вычислительных сервисов, таких как:
  + Amazon EC2,
  + AWS Fargate,
  + AWS Lambda и
  + локальных серверов.
* AWS CodeDeploy упрощает быстрый выпуск новых возможностей, помогает избежать простоев во время развертывания и берет на себя все сложности при обновлении приложений.
* AWS CodeDeploy можно использовать для автоматизации развертывания программного обеспечения, устраняя необходимость в подверженных ошибкам ручных операциях.
* Сервис масштабируется в соответствии с вашими потребностями.

**AWS CodePipeline**

* AWS CodePipeline — это сервис **непрерывной интеграции и доставки**, позволяющий быстро и надежно выполнять обновления приложений и инфраструктуры.
* CodePipeline выполняет **компоновку**, **тестирование** и **развертывание** вашего кода каждый раз **при внесении в него изменений**, используя определяемые вами модели процесса выпуска.
* Это позволяет быстро и надежно осуществлять доставку различных возможностей и обновлений.

**AWS CodePipeline**

* Это центральный инструмент для организации DevOps на Amazon AWS.
* Взаимодействует с сервисами:
  + CodeCommit;
  + CodeBuild;
  + CodeDeploy.
  + запуская их в нужной последовательности и, собственно, **обеспечивая автоматизацию DevOps-процессов**
* Позволяет;
  + организовывать ветвления процессов,
  + запускать сторонние сервисы (например, для тестирования),
  + делать параллельные ветки,
  + запрашивать подтверждение (*Approval Actions*) перед запуском следующего этапа.

**AWS CodeStar**

* Сервис, по сути дублирующий **CodePipeline.**
* С помощью AWS CodeStar можно:
  + настроить весь набор инструментов для непрерывной доставки ПО за считаные минуты, чтобы быстрее начать выпуск кода.
* AWS CodeStar облегчает безопасную совместную работу команды.
* С помощью шаблонов проекта, предлагаемых AWS CodeStar, можно легко разрабатывать разнообразные приложения:
  + от веб-сайтов и интернет-приложений
  + до интерактивных сервисов и приложений с использованием технологий Amazon Alexa.
* Шаблоны проектов AWS CodeStar включают необходимый для начала работы код на поддерживаемых языках программирования, включая Java, JavaScript, PHP, Ruby, C# и Python.

**IaaC-сервисы Amazon AWS**

* Сервисы Amazon AWS, реализующие концепцию «**Инфраструктура как код**», это:
  + **Elastic Beanstalk**
  + **OpsWorks**
  + **CloudFormation**
* Чтобы сделать что-то быстро (и это не значит, что плохо), это какой-то стандартный функционал (например, простой сайт) — удобно использовать **Elastic Beanstalk**.
* Если это сложный проект с многочисленными вложенными элементами и серьёзными требованиям по сетевым настройкам — без использования **CloudFormation** не обойтись.
* Как нечто среднее — представляется использование **OpsWorks**, базирующегося на Chef.

**IAAC Сложных проектов**

* На сложных проектах обычно используется комбинация из всех трёх служб IaaC:
  + **CloudFormation** :
    - поднимает основную инфраструктуру (**VPC**, подсети, репозитории, создаёт нужные роли в **IAM** для доступа и т.д.),
    - после запускает **OpsWorks**-стек, который уже может гибко настроить внутреннюю составляющую запущенных виртуалок.
    - может поднять и стек для **Elastic Beanstalk** компонентов, чтобы разработчики с помощью расширений могли сами менять некоторые параметры работающего приложения (количество и тип используемых виртуалок, использование *Load Balancer* и т.д.) просто изменяя простой файл конфигурации в папке с кодом, когда применение изменений (в т.ч. в инфраструктуру приложения) происходит автоматически после коммита.

**Jenkins на AWS**

* Jenkins - это сервер автоматизации с открытым исходным кодом, который интегрируется с рядом сервисов AWS, таких как AWS CodeCommit, AWS CodeDeploy, Amazon EC2 Spot и Amazon EC2 Fleet.
* Вы можете использовать Amazon Elastic Compute Cloud (Amazon EC2) для развертывания приложения Jenkins на AWS за считанные минуты.
* Функционирующий конвейер создает полностью управляемую службу сборки, которая компилирует исходный код.
* Затем он создает артефакты кода, которые CodeDeploy может использовать для автоматического развертывания в производственной среде.
* Рабочий процесс развертывания начинается с размещения кода приложения в репозитории GitHub.
* Чтобы автоматизировать этот сценарий, добавлено управление исходным кодом в проект Jenkins в разделе «Исходный код».
* Выбран вариант GitHub, который по дизайну клонирует копию содержимого репозитория GitHub в локальном каталоге рабочей области Jenkins.
* На втором этапе процедуры автоматизации включается триггер для сервера Jenkins, используя параметр «Опрос SCM». Эта опция заставляет Дженкинс проверять настроенный репозиторий на предмет любых новых коммитов/изменений кода с указанной частотой.

**AWS X-Ray**

* AWS X-Ray помогает разработчикам **анализировать и отлаживать** распределенные приложения **в рабочей среде** и на этапе разработки, например приложения, созданные с использованием архитектуры микросервисов.
* С помощью X-Ray можно **оценить производительность** своего приложения и базовых сервисов, чтобы найти и устранить основную причину проблем производительности и ошибок.
* X-Ray обеспечивает **комплексное отслеживание запросов** и их маршрутов в приложении и показывает карту внутренних компонентов приложения.
* X-Ray используется для **анализа приложений на стадии разработки и развертывания**, от простых **трехступенчатых** вариантов до сложных приложений, состоящих из **тысяч микросервисов**.

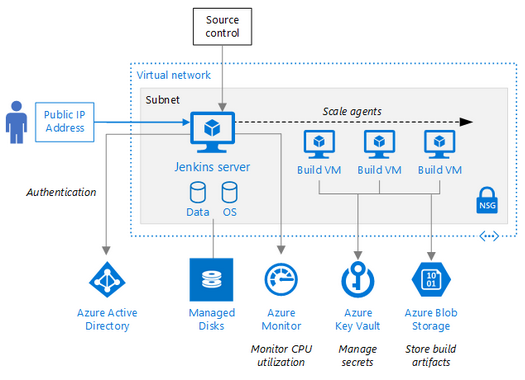
1. Реализация DevOPs в Azure.

**Azure:  
 Службы и Инструменты Devops**

**Azure Devops**

* В Azure DevOps входят:
  + **Azure Pipelines -** Непрерывная интеграция и развертывание, поддержка любых языков, платформ и облачных сервисов. Подключение к GitHub и любому репозиторию Git, непрерывное развертывание.
  + **Azure Boards -** Мощные инструменты контроля рабочего процесса: канбан-доски, журналы невыполненных работ, командные информационные панели и настраиваемые отчеты.
  + **Azure Artifacts -** Каналы пакетов Maven, npm и NuGet из открытых и закрытых источников.
  + **Azure Repos -** Закрытые облачные репозитории Git неограниченного объема для хранения файлов проекта Совместные запросы на включение, улучшенное управление файлами и многое другое.
  + **Azure Test Plans -** Комплексное решение для планирования и произвольного тестирования.

**Запуск сервера Jenkins в Azure**



* Для внедрения непрерывной интеграции и непрерывной доставки (CI/CD) для проектов программного обеспечения можно разместить развертывание Jenkins в Azure либо расширить имеющуюся конфигурацию Jenkins с помощью ресурсов Azure.
* Кроме того, можно использовать подключаемые модули Jenkins, чтобы упростить непрерывную интеграцию и непрерывную доставку приложений в Azure.

**Развитие Visual Studio Team Services (VSTS)**

* Azure DevOps – это результат развития Visual Studio Team Services (VSTS). Пользователи VSTS будут автоматически переведены на Azure DevOps. Существующие пользователи не потеряют в функциональности, однако приобретут больше возможностей выбора и контроля. Полная прозрачность и интеграция, отличительные особенности VSTS, сохранились. Службы Azure DevOps отлично сочетаются друг с другом.
* Перевод пользователей на использование Azure DevOps включает:
  + URL-адреса изменятся с abc.visualstudio.com на dev.azure.com/abc.
  + Будет включена переадресация с адреса visualstudio.com, поэтому вы не столкнетесь с битыми ссылками.
  + Поскольку служба изменилась, обновился и интерфейс. У новых пользователей он будет включен по умолчанию. Он станет основным и для существующих пользователей.
* Локальный сервис Team Foundation Server (TFS) будет по-прежнему обновляться, обновления будут аналогичны тем, что выпускаются для Azure DevOps.
* После выхода следующей версии TFS продукт получит новое название — Azure DevOps Server и будет обновляться в прежнем режиме.

1. Обзор существующих угроз информационной безопасности облачных вычислений. Угрозы безопасности по отношению к ВМ.

**Основные угрозы информационной безопасности в облачных инфраструктурах**

**CSA (Cloud Security Alliance)** представила список 12 угроз облачной безопасности, с которыми сталкиваются организации.

**Угроза 1: утечка данных**

* Из-за большого количества данных, которые сегодня часто переносятся в облака, площадки облачных хостинг-провайдеров становятся привлекательной целью для злоумышленников. При этом серьезность потенциальных угроз напрямую зависит от важности и значимости хранимых данных.
* Чтобы минимизировать риски и угрозы утечки данных, CSA рекомендует использовать **многофакторную аутентификацию при доступе** и **шифрование данных**.

**Угроза 2: компрометация учетных записей и обход аутентификации**

* Утечка данных зачастую является результатом небрежного отношения к механизмам организации проверки подлинности, когда используются слабые пароли, а управление ключами шифрования и сертификатами происходит ненадлежащим образом.
* Кроме того, организации сталкиваются с проблемами управления правами и разрешениями, когда конечным пользователям назначаются гораздо б*о*льшие полномочия, чем в действительности необходимо.

CSA рекомендует использовать механизмы:

* многофакторной аутентификации,
* одноразовые пароли,
* токены,
* смарт-карты,
* USB-ключи

**Угроза 3: взлом интерфейсов и API**

* От того, насколько хорошо проработаны механизмы контроля доступа, шифрования в API, зависит безопасность и доступность облачных сервисов. При взаимодействии с третьей стороной, использующей собственные интерфейсы API, риски значительно возрастают. Почему? Потому что требуется предоставлять дополнительную информацию, вплоть до логина и пароля пользователя.
* Слабые с точки зрения безопасности интерфейсы становятся узким местом в вопросах доступности, конфиденциальности, целостности и безопасности.

**CSA рекомендует**:

* организовать адекватный контроль доступа,
* использовать инструменты защиты и раннего обнаружения угроз.
* умение моделировать угрозы и находить решения по их отражению — достойная профилактика от взломов: - выполнять **проверку безопасности кода** и **запускать тесты на проникновение**.

**Угроза 4: уязвимость используемых инфраструктурных элементов систем**

* Уязвимость используемых систем — проблема, встречающаяся в **мультиарендных** облачных средах.
* Распространена ошибка, когда при использовании облачных решений в модели IaaS уделяется недостаточное внимание безопасности своих приложений, которые размещены в защищенной инфраструктуре облачного провайдера.
* Уязвимость виртуальных машин и самих приложений становится узким местом в безопасности корпоративной инфраструктуры.
* Рекомендации:
  + регулярное сканирование элементов инфраструктуры на выявление уязвимостей;
  + сегментация виртуальных машин;
  + сегментация баз данных;
  + самодиагностика виртуальных машин;
  + применение последних патчей для ОС и ПО виртуальных машин;
  + быструю реакцию на сообщения об угрозах безопасности.
  + защита от DDoS атак исходящих от ВМ, работающих в том же облаке.

**Угроза 5: кража учетных записей**

* Фишинг, мошенничество, эксплойты встречаются и в облачном окружении.
* Сюда добавляются угрозы в виде попыток манипулировать транзакциями и изменять данные.

Защита:

* Необходимо запретить «шаринг» учетных записей пользователей и служб между собой, а также обратить внимание на механизмы многофакторной аутентификации.
* Сервисные аккаунты и учетные записи пользователей необходимо контролировать, детально отслеживая выполняемые транзакции.
* Главное — обеспечить защиту учетных записей от кражи, рекомендует CSA.

**Угроза 6: инсайдеры-злоумышленники**

* Инсайдерская угроза может исходить от нынешних или бывших сотрудников, системных администраторов, подрядчиков или партнеров по бизнесу.
* Инсайдеры-злоумышленники преследуют разные цели, начиная от кражи данных до желания просто отомстить.
* В случае с облаком цель может заключаться в полном или частичном разрушении инфраструктуры, получении доступа к данным и пр..
* Системы, напрямую зависящие от средств безопасности облачного поставщика, — большой риск.
* CSA рекомендует:
  + позаботиться о механизмах шифрования и взять под собственный контроль **управление ключами шифрования**.
  + Не стоит забывать про **логирование**, **мониторинг** и **аудит** событий по отдельно взятым учетным записям.

**Угроза 7: целевые кибератаки**

* Злоумышленника, задавшегося целью установить и закрепить собственное присутствие в целевой инфраструктуре, не так легко обнаружить. (Развитая устойчивая угроза, или целевая кибератака)
* Для минимизации рисков и профилактики подобных угроз поставщики облачных услуг используют продвинутые средства безопасности.
* Но помимо современных решений, требуется понимание сущности и природы такого вида атак.
* CSA рекомендует:
  + проводить специализированное обучение сотрудников по распознаванию техник злоумышленника,
  + использовать расширенные инструменты безопасности,
  + уметь правильно управлять процессами,
  + знать о плановых ответных действиях на инциденты,
  + применять профилактические методы,
  + повышающие уровень безопасности инфраструктуры.

**Угроза 8: перманентная потеря данных**

* Облачные хостинг-провайдеры для соблюдения мер безопасности рекомендуют отделять пользовательские данные от данных приложений, сохраняя их в различных локациях.
* Не стоит забывать и про эффективные методы резервного копирования. Ежедневный бэкап и хранение резервных копий на внешних альтернативных защищенных площадках особенно важны для облачных сред.
* Кроме того, если клиент шифрует данные до размещения в облаке, стоит заранее позаботиться о безопасности хранения ключей шифрования.
* Как только они попадают в руки злоумышленнику, с ними становятся доступны и сами данные, потеря которых может быть причиной серьезных последствий.

**Угроза 9: недостаточная осведомленность**

* Организации, которые переходят в облако без понимания облачных возможностей, сталкиваются с рисками. Если, к примеру, команда разработчиков со стороны клиента недостаточно знакома с особенностями облачных технологий и принципами развертывания облачных приложений, возникают операционные и архитектурные проблемы.
* CSA напоминает о необходимости понимать функционирование облачных сервисов, предоставляемых поставщиком услуг. Это поможет ответить на вопрос, какие риски берет на себя компания, заключая договор с хостинг-провайдером.

**Угроза 10: злоупотребление облачными сервисами**

* Облака могут использоваться легитимными и нелегитимными организациями.
* Цель последних — использовать облачные ресурсы для совершения злонамеренных действий:
  + запуска DDoS-атак, отправки спама,
  + распространения вредоносного контента и т. д.
* Поставщикам услуг крайне важно уметь распознавать таких участников, для чего рекомендуется детально изучать трафик и использовать инструменты мониторинга облачных сред.

**Угроза 11: DDoS-атаки**

* Несмотря на то что DoS-атаки имеют давнюю историю, развитие облачных технологий сделало их более распространенными.
* В результате DoS-атак может сильно замедлиться или вовсе прекратиться работа значимых для бизнеса компании сервисов.
* Известно, что DoS-атаки расходуют большое количество вычислительных мощностей, за использование которых будет платить клиент.
* Несмотря на то что принципы DoS-атак, на первый взгляд, просты, необходимо понимать их особенности на прикладном уровне:
  + они нацелены на уязвимости веб-серверов и баз данных.
* Облачные поставщики, безусловно, лучше справляются с DoS-атаками, чем отдельно взятые клиенты. Главное — иметь план смягчения атаки до того, как она произойдет.

**Угроза 12: совместные технологии, общие риски**

* Уязвимости в используемых технологиях — достаточная угроза для облака.
* Поставщики облачных услуг предоставляют виртуальную инфраструктуру, облачные приложения, но если на одном из уровней возникает уязвимость, она влияет на все окружение.
* CSA рекомендует использовать стратегию «безопасности в глубину»:
  + внедрять механизмы многофакторной аутентификации,
  + системы обнаружения вторжений,
  + придерживаться концепции сегментирования сети,
  + принципа предоставления наименьших привилегий.

**Угрозы безопасности на основе виртуальной машины/гипервизора**

* По отношению к ВМ имеются следующие угрозы их безопасности:
  + Несанкционированное изменение файлов образов ВМ;
  + Кража ВМ;
  + Угон ВМ;
  + Атаки между ВМ;
  + Мгновенный пролом.

**Несанкционированное изменение файлов образов виртуальной машины**

* Виртуальные машины подвержены угрозам безопасности, как когда они работают, так и когда они выключены. Когда виртуальная машина выключена, она доступна в виде **файла образа виртуальной машины**.
* Этот файл образа подвержен нескольким угрозам безопасности:
  + например заражению вредоносным ПО.
  + Кроме того, при отсутствии соответствующих мер безопасности файлы образов виртуальных машин могут использоваться хакерами для создания новых неавторизованных виртуальных машин.
  + Также можно исправить файлы образов виртуальных машин, чтобы заразить виртуальные машины, создаваемые с использованием этих файлов.
* Безопасность ВМ может быть поставлена под угрозу даже во время ее миграции, при этом возможны такие атаки, как:
  + как перехват ВМ и несанкционированная модификация.
* Одна из техник, которые можно использовать для защиты файлов образов виртуальных машин, - это **зашифровать** их, когда они выключены или переносятся.

**Кража виртуальной машины**

* Кража виртуальной машины выполняется злоумышленником путем копирования или перемещение виртуальной машины несанкционированным образом.
* В основном это возможно из-за **неадекватного (не безопасного) управления** файлами виртуальных машин.
* Кража ВМ может оказаться фатальной, если украденная ВМ содержит конфиденциальные данные клиента.
* Один из способов ограничить кражу виртуальных машин - наложить на файлы виртуальных машин ограничения на копирование и перемещение. Такие ограничения эффективно привязывают виртуальную машину к определенной физической машине таким образом, что даже если есть несанкионированная копия машины, она не будет работать на любой другой физической машине. Виртуальная машина с таким уровнем ограничений копирования и перемещения не может работать на гипервизоре, установленном на любой другой физической машине.

**Побег ВМ**

* Помимо кражи ВМ, на уровне ВМ может возникнуть еще одна угроза, известная как «побег ВМ».
* Обычно виртуальные машины инкапсулируются в отдельный образ и изолированы друг от друга и от базового родительского гипервизора.
* В обычных сценариях для гостевой ОС и приложений, работающих на ней, нет механизма, позволяющего выйти за границы виртуальной машины и напрямую взаимодействовать с гипервизором.
* Процесс выхода из виртуальной машины и взаимодействия с гипервизором называется выходом из виртуальной машины.
* Поскольку гипервизор контролирует выполнение всех виртуальных машин, эта атака позволяет злоумышленнику получить контроль над любой другой виртуальной машиной.

**Атаки между виртуальными машинами**

* На одной физической машине (сервере виртуализации) как правило работают несколько виртуальных машин. Таким образом, если безопасность одной виртуальной машины скомпрометирована, существует очень большая вероятность того, что безопасность других виртуальных машин, работающих на той же физической машине, будет скомпрометирована.
* В одном сценариев злоумышленник **может скомпрометировать одну гостевую виртуальную машину**, которая затем может быть использована для атак на другие виртуальные машины, работающим на той же физической машине.
* Решение:
* Чтобы предотвратить возникновение таких сценариев, очень важно иметь **брандмауэры** и **системы обнаружения вторжений** в сети виртуальных машин, а также и на самих ВМ, которые могут обнаруживать и предотвращать вредоносную активность на уровне виртуальных машин.

**Мгновенные дыры (Instant-on Gaps)**

* Виртуальные машины имеют некоторые уязвимости, которых нет на физических машинах.
* В основном это связано с технологиями, которые используются для снабжения их ресурсами, использования и отключения ВМ.
* Иногда эти циклы повторяются очень часто. Такая частая активация и деактивация виртуальных машин может создать **проблемы** для поддержания их систем безопасности в состоянии полного **обновления**. В результате в защите ВМ возникают «**мгновенные дыры**», что ведет к повышению опасности «взлома» ВМ.
* Также существует вероятность того, что новые виртуальные машины могут быть клонированы и созданы из виртуальных машин, которые имеют уязвимости.
* Одним из решений этой проблемы может быть **выделенная виртуальная машина безопасности** на каждой физической машине для автоматического обновления всех виртуальных машин, работающих на этой физической машине, всеми последними обновлениями безопасности.

**Угон гипервизора (Hyperjacking)**

* Установка на физический сервер мошеннического гипервизора, который может полностью контролировать базовый физический сервер.
* Это достигается с помощью руткита - вредоносной программы, которая устанавливается до полной загрузки гипервизора на физическом сервере.
* Используя мошеннический гипервизор, злоумышленник может:
  + запускать неавторизованные приложения в гостевой ОС, при этом ОС не осознает наличие таких приложений.
  + контролировать взаимодействие между виртуальными машинами и базовым физическим сервером.
* Регулярные меры безопасности неэффективны против этого мошеннического гипервизора, потому что:
  + Гостевая ОС не знает о том, что базовый сервер был атакован.
  + Приложения антивируса и брандмауэра не могут обнаружить присутствие мошеннического гипервизора, поскольку он установлен непосредственно на самом сервере.
* Меры против гиперджекинга включают в себя:
  + Аппаратная поддержка безопасного запуска гипервизора, предотвращающего запуск вредоносных программ уровня руткитов. Это потребует разработки и использования TCB для поддержки гипервизора на аппаратном уровне.
  + Сканирование деталей на уровне оборудования для оценки целостности гипервизора и определение наличия мошеннического гипервизора. Это сканирование может включать в себя проверку состояния памяти, а также регистрацию ЦП.