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

Распределенная информационная система — это совокупность элементов, которые географически удалены друг от друга, но для пользователя выглядят как единое целое.

• **Коллекция автономных вычислительных элементов:** Современные системы состоят из различных узлов, которые могут действовать независимо друг от друга. Это приводит к двум важным последствиям:

◦ **Независимость времени:** у каждого узла свое представление о текущем времени, что требует специальной синхронизации и координации.

◦ **Групповое управление:** узлы могут объединяться в открытые или закрытые группы для эффективного управления безопасностью.

• **Оверлейные (наложенные) сети:** Это логические сети, которые образуются «поверх» физических соединений. Они всегда связны (между любыми двумя узлами есть путь) и бывают двух видов: **структурные** (узлы знают своих соседей) и **неструктурные** (соседи выбираются случайно).

• **Единая согласованная система:** Несмотря на то что компоненты автономны, программное обеспечение скрывает различия между ними. Для пользователя РИС — это «черный ящик». Это создает ощущение **прозрачности**, но порождает проблему: отказ одного скрытого узла может сделать непригодным к использованию весь компьютер пользователя.

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

Понятие РИС появилось благодаря двум технологическим новинкам: мощным процессорам и высокоскоростным компьютерным сетям.

Эволюция аппаратных платформ и систем:

1. **1950–1970-е:** Эпоха мэйнфреймов (больших ЭВМ).
2. **1960–1980-е:** Появление недорогих мини-компьютеров.
3. **1980–2000-е:** Бум портативных компьютеров и мобильных устройств.
4. **1990-е – настоящее время:** Развитие кластеров, грид-систем и облачных вычислений.

Хронология развития технологий:

- **1960-е:** Появление ОС с межпроцессным взаимодействием и сети ARPANET.
- **1970-е:** Распространение локальных сетей (Ethernet).
- **1980-е:** Распределенные вычисления становятся активно изучаемой темой.
- **2000-е:** Корпоративные системы повсеместно переходят на распределенную модель.

Современные РИС сформировались на базе двух классов систем: **HPC** (высокопроизводительные вычисления для науки) и **HTC** (системы высокой пропускной способности для бизнеса).

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

Классификация РИС:

• **По архитектуре:** клиент-серверные, многоуровневые (N-tier), одноранговые (P2P), кластерные.

• **По территории:** локальные, городские, глобальные.

• **По степени автоматизации:** автоматизированные и автоматические.

• **По способу обработки:** информационно-справочные и вычислительные.

Преимущества:

• Доступ к физически удаленным ресурсам.

• Более высокая производительность за счет параллельной работы.

• Высокая надежность за счет дублирования компонентов.

Недостатки:

• Сложность разработки, отладки и контроля удаленных элементов.

• Увеличенное время реакции системы (задержки сети).

• Дополнительные усилия по обеспечению безопасности.

Особенности современных РИС: Они крайне сложны структурно, включают в себя множество подсистем и часто требуют интеграции со старыми (унаследованными) приложениями. Современные системы многоплатформенны и рассчитаны на длительный срок существования.

4. Состав РИС. Виды обеспечения РИС. Корпоративные РИС.

Состав РИС включает в себя пять элементов:

1. **Хосты (компьютеры):** вычислительные узлы.
2. **Сеть:** средства коммуникации.
3. **Приложения:** программное обеспечение.
4. **Данные:** информация в хранилищах.
5. **Пользователи.**

Виды обеспечения:

- **Информационное:** структуры данных и способы их передачи.
- **Техническое:** «железо».
- **Математическое и программное:** алгоритмы и системное/прикладное ПО.
- **Организационное:** правила работы людей с системой.
- **Правовое:** нормативная база.

Корпоративные РИС: Это информационные системы организаций, которые используют клиент-серверные модели, распределенную обработку транзакций и методы интеграции различных бизнес-приложений.

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

При создании РИС разработчики стремятся достичь следующих свойств:

• **Доступность (Готовность):** Свойство системы находиться в рабочем состоянии в любой момент времени. Она измеряется как вероятность того, что система выполнит свою функцию по требованию пользователя.

• **Прозрачность:** Система кажется пользователю единым целым. Она может скрывать местоположение ресурса, факт его перемещения, наличие копий (репликацию) или возникновение сбоев.

• **Открытость:** Использование открытых стандартов и протоколов. Это обеспечивает **интероперабельность** (способность разных систем взаимодействовать) и **переносимость** приложений.

• **Гибкость:** Способность системы подстраиваться под новые нужды. Бывает программной (замена версий ПО), аппаратной (изменение числа серверов), структурной или логической (изменение бизнес-процессов).

• **Масштабируемость:** Способность системы расти (по числу пользователей или географии) без потери производительности.

◦ **Вертикальное масштабирование:** замена компонентов на более мощные.

◦ **Горизонтальное масштабирование:** добавление новых узлов в систему.

6. Типы распределенных систем.

Современные распределенные системы (РС) сформировались в результате развития двух основных классов систем: **HPC** (высокопроизводительные вычисления для науки) и **HTC** (системы высокой пропускной способности для бизнеса).

На сегодняшний день выделяют три основных типа:

1. **Системы высокопроизводительных распределенных вычислений (HPC):** используются там, где нужны огромные вычислительные мощности (наука, инженерия). Первоначально это были суперкомпьютеры, которые сейчас заменяются **кластерами** — наборами однородных компьютеров, соединенных быстрой сетью.
2. **Системы распределенных параллельных вычислений (информационные системы):** это прежде всего **корпоративные системы**, где важно управление бизнес-процессами, распределенная обработка транзакций и интеграция различных приложений внутри организации.
3. **Повсеместные (всеобъемлющие) системы:** это мобильные и встраиваемые устройства, системы слежения с датчиками, а также **Интернет вещей (IoT)**. Их особенность в том, что они часто действуют автономно и «ненавязчиво» для пользователя.

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

Grid-системы (Грид) — это архитектура, которая координирует ресурсы разных организаций для совместного использования.

• **Идея:** пользователь может запустить сложную задачу с любого устройства, а ресурсы для вычислений будут предоставлены автоматически из «федерации» серверов.

• **Аналогия:** это похоже на **электрическую сеть** — нам не важно, какая станция выработала ток, мы просто включаем прибор в розетку.

• **Архитектура:** строится на базе стандарта **OGSA** (открытая архитектура грид-сервисов), которая опирается на сервисно-ориентированный подход (SOA).

Облачные вычисления (Cloud Computing) — это развитие идеи вычислений «как услуги» (Utility Computing), где ресурсы предоставляются по запросу через Интернет с оплатой по факту использования. Согласно модели **NIST**, облака имеют 5 ключевых характеристик:

1. **Самообслуживание по запросу:** пользователь сам получает ресурсы без участия провайдера.
2. **Широкополосный доступ:** работа через интернет с любого устройства.
3. **Объединение ресурсов в пулы:** ресурсы провайдера динамически распределяются между всеми клиентами.
4. **Быстрая эластичность:** возможность мгновенно увеличивать или уменьшать объем услуг.
5. **Измеряемое обслуживание:** прозрачный контроль и оплата только за потребленное

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

Для управления распределенными системами используются три основных подхода:

• **Распределенные ОС:** это «сильносвязанные» системы, которые управляют множеством процессоров или одинаковых компьютеров так, будто это одна машина. Они обеспечивают **высочайшую прозрачность**, но плохо масштабируются и являются закрытыми.

• **Сетевые ОС:** это «слабосвязанные» системы, где каждый компьютер имеет свою ОС. Они позволяют пользователям обращаться к удаленным ресурсам, но их **прозрачность низкая** (пользователь видит, что работает с другим узлом).

• **ПО промежуточного слоя (Middleware):** это специальный программный слой, который «накладывается» поверх сетевых ОС. Его главная задача — **скрыть различия** между разными платформами и операционными системами, делая систему прозрачной для приложений.

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

В классической модели **клиент-сервер** процессы делятся на две группы: **серверы** (предоставляют услуги) и **клиенты** (запрашивают их).

Для эффективной работы приложения разделяют на **три логических уровня**:

1. **Уровень представления (UI):** интерфейс, через который взаимодействует человек или другая программа.
2. **Уровень обработки (Бизнес-логика):** «сердце» системы, где выполняются основные алгоритмы.
3. **Уровень данных:** источник информации (базы данных, файлы).

Варианты архитектуры:

- **Тонкий клиент:** почти вся обработка и хранение данных происходят на сервере, клиент отвечает только за интерфейс. Это удобно для мобильных устройств.
- **Толстый клиент:** часть логики приложения (обработка данных) переносится на сторону клиента. Это усложняет администрирование и обновление программ.
- **Многоуровневые (N-tier) системы:** функции обработки данных выделяются в отдельный модуль (Middleware), что повышает защищенность и гибкость управления.

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

Организация системы — это то, как программное обеспечение распределено между узлами. Она бывает:

• **Логическая (программная):** состав программных компонентов и то, как они общаются.

• **Физическая (системная):** реальное размещение этих компонентов на конкретных компьютерах.

Прозрачность и архитектура: Чтобы система казалась пользователю единым целым, необходимо четко разделять само приложение и базовую платформу (железо + ОС). Это разделение выполняется с помощью **Middleware (промежуточного уровня)**. Выбор того, где именно разместить этот слой, определяет программную архитектуру системы.

Виды системной архитектуры:

1. **Централизованная:** классический клиент-сервер.
2. **Децентрализованная (P2P):** все узлы системы равны и действуют одновременно и как клиенты, и как серверы.
3. **Гибридная:** сочетание элементов централизованного и одноранговового подходов.

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

Архитектура системы — это её структура, описанная в терминах отдельных компонентов (сущностей) и взаимоотношений между ними. Выбор архитектуры является ключевым решением, так как именно она определяет будущую масштабируемость, надежность, открытость и безопасность системы.

- **Архитектурный стиль** определяет, как именно элементы системы будут взаимодействовать, как будет происходить обмен данными и как они совместно сформируют единую распределенную систему.
- **Программный компонент** — это модульная единица программного обеспечения, которая имеет полностью определенный интерфейс. Главное свойство компонента — его **заменяемость**: его можно заменить новым (при условии сохранения того же интерфейса) в любой момент, даже во время работы системы.
- **Интерфейс** — описание параметров, необходимых для обращения к компоненту.
- **Коннектор** — механизм, который обеспечивает связь и координацию между компонентами (например, удаленный вызов процедур или обмен сообщениями).

Основные виды архитектурных стилей:

1. **Многоуровневые (layered):** компоненты делятся по функциям (например: представление, обработка, данные).
2. **Объектные (object-based):** система строится из взаимодействующих распределенных объектов.
3. **Компонентные (component-based):** развитие объектного стиля с явным описанием всех зависимостей.
4. **Сервисно-ориентированные (SOA):** объединение различных сервисов (часто Web-сервисов).
5. **Ресурсо-центрированные:** система рассматривается как коллекция ресурсов, управляемых своими компонентами.
6. **На основе событий (event-based):** коммуникация происходит через публикацию и подписку на события.

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

Сущности (строительные блоки) РИС рассматриваются в двух аспектах: **системном** и **проблемном**.

• **Системный аспект:** определяет физическую реализацию. В современных системах это **процессы** или **потоки** операционной системы, которые потребляют реальные ресурсы (память, процессор) и имеют сетевую точку подключения (IP-адрес, порт).

• **Проблемный аспект:** определяется предметной областью (бизнес-логикой). Здесь используются абстракции:

◦ **Объекты:** представляют объекты реального мира. Доступ к ним идет через интерфейсы, описанные на языке **IDL**.

◦ **Объектные компоненты:** в отличие от объектов, они не только дают интерфейс «наружу», но и четко указывают, какие другие интерфейсы нужны им для работы. Это позволяет сторонним разработчикам создавать совместимые части системы.

◦ **Сервисы:** сущности, инкапсулирующие поведение. **Web-сервисы** используют открытые стандарты (XML, HTTP), что позволяет легко объединять их в сложные системы разных организаций (B2B).

◦ **Ресурсы:** рассматриваются как индивидуально управляемые элементы, что упрощает построение очень крупных Web-систем.

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

Взаимодействие между сущностями бывает трех видов:

1. **Межпроцессные коммуникации (IPC / прямые):** Это низкоуровневый способ общения напрямую через сетевые протоколы (программирование сокетов TCP/IP) или простой обмен сообщениями.
2. **Удаленный вызов (Remote Call):** Самый популярный метод. Он имитирует вызов обычной функции, но на другом компьютере. Сюда входят:
   - **RPC (Remote Procedure Call):** вызов удаленной процедуры.
   - **RMI (Remote Method Invocation):** вызов метода удаленного объекта.
3. **Непрямые коммуникации (indirect):** Взаимодействие через посредника (третью сторону). Это дает:
   - **Пространственное разъединение:** отправитель может не знать, кто получит сообщение.
   - **Временное разъединение:** отправитель и получатель не обязаны быть в сети одновременно.
4. **Групповые коммуникации:** Сообщение доставляется сразу набору получателей (парадигма «один-ко-многим»). Отправитель шлет данные на **идентификатор группы**, а система сама находит всех её участников.

14. 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).

Эти модели позволяют сделать систему более гибкой и устойчивой к сбоям отдельных узлов.

• **Системы публикации-подписки (publish-subscribe):** Издатели (publishers) генерируют события, а подписчики (subscribers) получают только те данные, на которые они подписались через промежуточный сервис. Бывают основаны на **теме** (topic-based) или на **содержимом** (content-based).

• **Очередь сообщений (Message queues):** Модель «точка-точка». Отправитель кладет сообщение в очередь, а получатель забирает его оттуда, когда сможет.

• **Пространство записей / кортежей (Tuple spaces):** Модель «генерирующих коммуникаций» (например, Linda). Процессы кладут структурированные данные (кортежи) в общее постоянное хранилище. Другие могут читать или удалять их по шаблону. Это позволяет общаться программам, которые никогда не работали одновременно.

• **Распределенная разделяемая память (DSM):** Создает иллюзию общей оперативной памяти для процессов на разных компьютерах. Программист пишет код так, будто он работает с локальной памятью, а инфраструктура сама синхронизирует данные между узлами.

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

Роль сущности — это фундамент архитектуры.

Архитектура клиент-сервер (централизованная):

• Процессы делятся на две группы: **серверы** (предоставляют услугу) и **клиенты** (запрашивают её).

• На клиенте и сервере обычно используется разное программное обеспечение.

• **Проблемы:** сложно масштабировать, при отказе сервера вся система перестает работать («бутылочное горлышко»).

Архитектура Peer-to-Peer (P2P / децентрализованная):

• Все процессы (узлы) равны. Каждый узел одновременно является и клиентом, и сервером (симметричное взаимодействие).

• **Преимущества:** высочайшая масштабируемость, самоорганизация (узлы могут свободно входить в сеть и выходить из неё) и отсутствие единой точки отказа.

• В P2P часто используются **оверлейные сети** (логические связи поверх физических), которые бывают **структурированными** (строгая топология, например, кольцо для быстрого поиска) и **неструктурированными** (связи случайны).

Аналогия:

• **Клиент-сервер** — это **лекция в аудитории**: если лектор (сервер) замолчит, занятия прекратятся для всех студентов (клиентов).

• **P2P** — это **вечеринка**, где все общаются со всеми: если один человек уйдет, беседа в других группах не прекратится, а информация всё равно разойдется по всей комнате.

• **Публикация-подписка** — это **газета**: редакция (издатель) не знает каждого читателя лично, а читатель (подписчик) получает номер в свой почтовый ящик, даже если он проспал момент доставки.

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

Системная архитектура — это фактическая реализация распределенной системы, которая определяет, как именно программные компоненты размещаются на реальных физических машинах. Если программная организация — это логическая схема, то системная архитектура — это «физическая карта» того, где и что работает. Правильное размещение критически влияет на **производительность**, **надежность** и **безопасность** всей системы.

Основные стратегии размещения логических сущностей:

1. **Отображение служб на несколько серверов:** Служба может работать как несколько процессов на разных серверах. Например, браузер пользователя может перемещаться между разными веб-серверами, получая ресурсы от каждого из них.
2. **Кэширование:** Это создание хранилища для последних полученных данных. Примером является **кэширующий веб-прокси**, который сохраняет копии страниц, чтобы не запрашивать их каждый раз из интернета.
3. **Использование мобильного кода:** Часть программы (например, **JavaScript** или Java-апплеты) передается с сервера на клиент и выполняется там. Это ускоряет реакцию системы, но требует мощностей от устройства пользователя.
4. **Мобильные программные агенты:** Это полноценные программы (код + данные), которые «путешествуют» по сети между машинами, выполняют там задачи и возвращаются с результатом. Их используют для обслуживания ПО в компаниях или для сравнения цен на разных сайтах.

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

В **многоуровневой архитектуре (layered)** компоненты организованы вертикально. Основное правило: компонент верхнего уровня может вызывать функции только нижележащего уровня, но не наоборот.

Эволюция уровней:

• **1960–70-е годы:** Приложения были **одноуровневыми** (терминальные системы).

• **С появлением баз данных:** Сформировались **три классических уровня**: пользовательский интерфейс, обработка и данные.

• **Современная архитектура:** Уровень приложения разделился на подуровни: **представление**, **сервер приложений**, **модель данных** и **перманентность** (сохранение объектов).

Типовая архитектура современного корпоративного приложения:

1. **UI (User Interface):** Уровень взаимодействия с пользователем или другой программой.
2. **AL (Application Layer):** Уровень приложения, который координирует работу объектов.
3. **DL (Domain Logic):** «Сердце» системы, где сосредоточена вся бизнес-логика.
4. **Infrastructure:** Техническая поддержка, сохранение данных и обмен сообщениями.

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

Трехзвенная архитектура включает в себя **браузер (клиент)**, **сервер приложений** и **базу данных**.

Методы расширения функциональности на стороне сервера:

• **CGI (Common Gateway Interface):** Первый стандарт для программ, работающих с веб-сервером.

◦ *Плюс:* Высокая переносимость.

◦ *Минус:* Для каждого запроса создается новый процесс, что медленно и сильно нагружает сервер.

• **FastCGI:** Усовершенствованный вариант, где программа-партнер загружается один раз и работает постоянно в фоновом режиме.

◦ *Плюс:* Гораздо быстрее и экономнее, чем обычный CGI.

• **API-интерфейсы (динамические библиотеки):** Код встраивается прямо в память веб-сервера как библиотека (DLL).

◦ *Плюс:* Самая высокая производительность.

• **Серверы приложений (например, Cold Fusion):** Веб-сервер и логика приложения объединяются в один мощный продукт.

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

Методы расширения функциональности на стороне клиента:

1. **Вспомогательные программы (Helpers):** Отдельные программы на ПК пользователя. Браузер сначала скачивает файл целиком, а потом запускает «хелпер» для его открытия.
2. **Подключаемые модули (Plug-ins):** Тесно встроены в браузер. Например, **ActiveX**. Могут показывать данные (видео или звук) еще до того, как файл скачался до конца.
3. **Java-апплеты:** Скомпилированные программы, которые скачиваются и запускаются браузером. Они безопасны, так как каждая команда проверяется перед выполнением, и кроссплатформенны.
4. **Сценарии (JavaScript, VBScript):** Код встроен прямо в текст веб-страницы. Они идеально подходят для проверки того, правильно ли пользователь заполнил поля в форме, прежде чем отправлять её на сервер.

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

Шаблон с тонким клиентом: Подразумевает, что почти вся логика и обработка данных находятся на сервере, а клиент отвечает только за интерфейс. Это позволяет использовать простые устройства, например смартфоны. Технически это часто реализуется через **VNC** (виртуальные сетевые вычисления).

Шаблон с использованием прокси-сервера (посредника): В памяти клиентской машины создается «объект-заместитель» (прокси), который выглядит точно так же, как удаленный объект на сервере. Это создает **локальную прозрачность**: программист пишет код так, будто работает с локальным объектом, а прокси сам передает команды на сервер. Также прокси используются для кэширования и репликации данных.

Шаблон с использованием брокеров: Применяется, когда система состоит из множества сервисов. **Брокер** выступает посредником: приложения посылают ему запрос, а он сам находит нужный сервис, передает ему данные и возвращает ответ. Это упрощает связи в системе.

Шаблон с использованием рефлексии: Модель, позволяющая системе «изучать» саму себя.

• **Интроспекция:** система динамически обнаруживает свои свойства (например, какие методы есть у компонента).

• **Интерцессия:** система может сама менять свою структуру или поведение прямо во время работы. Это часто используется в сложном промежуточном ПО для настройки системы под текущие условия.

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

По мере роста сложности распределенных систем (особенно тех, что строятся на Web-сервисах), разработчики столкнулись с проблемой: обеспечивать связь между огромным количеством разных компонентов стало слишком трудно.

Ресурсо-центрическая архитектура — это альтернативный подход, где система рассматривается не как набор программ, а как **коллекция ресурсов**.

• **Суть:** Каждым ресурсом индивидуально управляет свой компонент или сервис.

• **Web-сервисы RESTful:** Это современный способ реализации такой архитектуры. В этой модели всё, к чему можно обратиться (файл, запись в базе данных, изображение), считается ресурсом,.

22. 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.

REST (Representational State Transfer) — это стиль проектирования, который делает взаимодействие в сети простым и стандартным.

• **Идентификация (URI):** Согласно стандартам W3C, каждый Web-сервис и каждый ресурс идентифицируются с помощью уникального адреса — **URI** (Uniform Resource Identifier).

• **Использование HTTP-методов:** REST-сервисы явно используют стандартные команды протокола HTTP (например, GET для получения данных, PUT для загрузки или изменения, DELETE для удаления),.

• **Состояние (Stateless):** Одной из ключевых идей является работа **без сохранения состояния** на стороне сервера (stateless). Это означает, что каждый запрос от клиента должен содержать всю информацию, необходимую для его понимания, а сервер не должен «помнить» о предыдущих запросах. Это значительно упрощает масштабирование системы.

• **Форматы данных:** Для обмена информацией Web-сервисы чаще всего используют **XML** (как указано в определении W3C) или другие структурированные форматы.

• **Сравнение REST и SOAP:**

◦ **SOAP** — это более строгий протокол, основанный на обмене XML-сообщениями. Он часто используется в сложных корпоративных системах для интеграции приложений (B2B),.

◦ **REST** — более легкий подход, который лучше подходит для облачных систем и масштабируемых Web-приложений,. Например, сервис Amazon S3 поддерживает оба варианта (REST и SOAP), но REST считается более простым для доступа из браузера,.

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

С ростом распределенных систем стало важно сделать процессы как можно менее зависимыми друг от друга. В архитектуре **Публикация/Подписка** вводится строгое разделение между теми, кто отправляет данные (**Издатели**), и теми, кто их получает (**Подписчики**),.

Способы координации в системе: Для того чтобы части системы работали вместе («склеивались»), используются два вида координации,:

1. **Временная:** должны ли процессы работать одновременно?
2. **Ссылочная (адресная):** должен ли отправитель знать точный адрес получателя?

Варианты архитектур P/P:

- С прямой координацией.
- Через почтовый ящик.
- На основе событий (через общую шину сообщений),.
- На основе разделяемых данных (пространство кортежей),.

Основная проблема: Главная сложность — это **эффективность и масштабируемость** процесса сравнения подписок с уведомлениями. Когда датчиков (источников событий) миллионы, системе очень трудно мгновенно понять, кому именно из тысяч подписчиков нужно отправить конкретное уведомление,.

24. Архитектура систем публикации/подписки с прямой координацией.

Это самая «строгая» модель взаимодействия.

• **Временная координация:** Существует. Оба процесса (издатель и подписчик) должны быть **запущены и работать одновременно** в момент передачи данных.

• **Ссылочная координация:** Существует. Издатель должен иметь **явную ссылку** на подписчика (его имя или сетевой идентификатор), чтобы знать, кому отправить уведомление

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

Эта модель более гибкая, так как процессы меньше зависят друг от друга во времени.

• **Координация по времени отсутствует:** Издатель и подписчик могут работать не синхронно. Отправитель периодически просыпается, кладет сообщение в «почтовый ящик» и может завершить работу. Подписчик также периодически проверяет свой ящик и забирает письма, когда ему удобно.

• **Ссылочная координация существует:** Хотя процессы не обязаны работать одновременно, отправитель всё равно должен знать **адрес почтового ящика** того процесса, которому он шлет информацию.

26. 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.

В этой модели взаимодействие между программами происходит не напрямую, а через **общую шину сообщений** (событий).

• **Варианты архитектуры:**

1. **На основе тем (topic-based):** каждое событие помечается определенной «темой» (ключевым словом). Подписчик говорит системе: «Я хочу получать всё по теме "Погода"», и промежуточное ПО фильтрует сообщения по этому признаку.

2. **На основе содержимого (content-based):** это более сложный вариант. Здесь подписчик задает условия, похожие на поисковые запросы (например, «присылай уведомление, только если температура выше 30 градусов и влажность ниже 10%»).

• **Принцип обмена данными через шину:** Издатель просто выбрасывает событие в «шину». Система (промежуточное ПО) сама сравнивает описание этого события с подписками всех пользователей. Если условия совпадают, система либо присылает подписчику само уведомление вместе с данными, либо просто сообщает, что событие произошло, а данные подписчик забирает позже из другого хранилища.

• **Главная проблема:** чем сложнее условия подписки (контентная фильтрация), тем труднее системе быстро обрабатывать миллионы запросов и уведомлений, что ограничивает её масштабируемость.

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

Распределенность в информационных системах реализуется двумя способами: **вертикально** и **горизонтально**.

- **Вертикальная распределенность (Клиент-сервер):** Приложение разбивается на иерархические уровни (интерфейс, логика, данные), и каждый уровень живет на своем узле.
  - *Минусы:* если сервер «упадет», работа встанет; сложно бесконечно расширять один мощный сервер.
- **Горизонтальная распределенность (P2P / Одноранговые системы):** Все узлы системы равны. Каждый компьютер (пир) одновременно является и клиентом (запрашивает данные), и сервером (отдает свои ресурсы другим).

Задачи, решаемые P2P:

1. **Распределение затрат:** компания-владелец не платит за огромные серверы, так как ресурсы предоставляют сами пользователи.
2. **Объединение мощностей:** можно объединить миллионы домашних ПК для сложных научных вычислений или хранения гигантских объемов файлов.
3. **Анонимность:** в сетях P2P труднее отследить пользователя, так как нет центрального сервера, который знает всё обо всех.

Принципы построения P2P:

- **Самоорганизация:** узлы сами находят друг друга, подстраиваются под включение или выход участников из сети.
- **Децентрализация:** нет «главного», который управляет поведением всех узлов.
- **Симметрия:** функции клиента и сервера выполняются каждым узлом одновременно.

28. Классификация P2P по степени централизации.

Хоть P2P и считаются свободными сетями, они делятся на три вида по тому, как в них управляются процессы:

1. **Централизованные:** Есть группа «управляющих узлов», которые отвечают за вход в сеть (аутентификацию) и хранят адресную книгу (индекс), где лежат файлы. Пример — старый Napster: вы спрашиваете у центра, где файл, он дает адрес, а скачиваете вы уже напрямую у соседа.
2. **Децентрализованные:** Центра нет вообще. Каждый узел знает только своих ближайших «логических соседей». Поиск идет «шаг за шагом» от одного соседа к другому. Это самые надежные, но медленные в поиске системы.
3. **Гибридные:** Самые популярные сегодня. В них выделяются **супер-узлы** (более мощные компьютеры с хорошим интернетом), которые помогают «слабым» узлам быстрее находить данные.

29. Гибридные архитектуры P2P систем. Системы с граничными серверами. Архитектура bittorrent.

Системы с граничными серверами: Эти серверы ставятся на «границе» сети интернет-провайдера. Они кэшируют (копируют) популярный контент из интернета, чтобы пользователи внутри сети получали его мгновенно. Это широко применяется в «Интернете вещей» (IoT).

Архитектура BitTorrent: Это пример гибридной системы, основанной на сотрудничестве.

• **Торрент-файл:** содержит информацию о частях файла и адрес **трекера**.

• **Трекер:** это сервер-координатор, который знает, у каких пользователей сейчас есть нужные части файла, и дает их адреса вашему компьютеру.

• **Обмен данными:** получив адреса, ваш компьютер общается напрямую с другими пользователями (**пирами** и **сидами**), минуя трекер.

• **Современный этап:** сегодня торренты могут работать и без трекера, используя технологию **DHT** (распределенные таблицы хэшей), где адреса хранятся самими пользователями.

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

Распределенный реестр (Distributed Ledger) — это база данных, копии которой хранятся одновременно на всех узлах сети P2P и синхронизируются по специальному алгоритму. В ней нет центрального администратора.

Виды распределенных реестров:

1. **Публичные:** открыты для всех, любой может скачать базу и участвовать в изменениях (например, Биткоин).
2. **Частные:** управляются одной организацией, доступ строго ограничен.
3. **Федеративные (Консорциумы):** управляются группой заранее выбранных узлов (например, банковским альянсом).

Пример использования: В **Эстонии** технология распределенного реестра используется в системе **KSI** для защиты медицинских записей граждан. Сами медицинские данные в реестре не лежат (для приватности), но там хранятся их «отпечатки» (хэш-значения). Если кто-то попытается тайно изменить историю болезни в государственной базе данных, «подпись» в реестре не совпадет, и система мгновенно обнаружит взлом.

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

Одноранговые (P2P) системы различаются по тому, как в них организованы связи между узлами (наложенная сеть).

• **Структурированные P2P-системы:** В таких сетях узлы соединяются по строгой математической схеме — **топологии** (например, в виде кольца, двоичного дерева или решетки). Протокол гарантирует, что любой ресурс будет найден эффективно, даже если он встречается в сети очень редко.

• **Индекс ресурса:** В структурированных сетях используется **бессемантический индекс**. Это значит, что каждому файлу или ресурсу присваивается уникальный числовой **ключ** (результат работы хэш-функции от его значения).

• **Распределенная хэш-таблица (DHT):** Это способ описания системы, где каждый узел получает свой уникальный идентификатор из того же диапазона, что и ключи данных. Каждый узел становится «ответственным» за хранение данных с определенным набором ключей.

• **Поиск узла и маршрутизация:** Чтобы найти файл, система выполняет операцию node = lookup(key). Поиск идет не наугад, а путем **маршрутизации**: запрос передается соседу, чей идентификатор математически ближе всего к искомому ключу.

• **Маршрутизация документов:** Ресурс (или ссылка на него) сохраняется на том узле, чей идентификатор наиболее похож на идентификатор самого документа. При поиске запрос просто транслируется к узлу с максимально похожим ID.

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

В таких сетях (например, Gnutella) нет строгих правил размещения данных, а топология связей случайна (случайный граф). Каждый узел знает только своих непосредственных соседей.

Методы поиска:

1. **Метод затопления (flooding):** Узел рассылает запрос всем своим соседям. Если у них нет данных, они пересылают запрос своим соседям и так далее. Чтобы сеть не «утонула» в запросах, используется **TTL (Time to Live)** — время жизни запроса. На каждом шаге TTL уменьшается, и при достижении нуля запрос удаляется.
2. **Метод случайного блуждания (random walk):** Узел выбирает только одного случайного соседа и спрашивает его. Если данных нет, тот выбирает своего случайного соседа. Это создает гораздо меньше трафика, но поиск может длиться долго. Для ускорения можно запустить несколько таких «блужданий» одновременно.
3. **Поиск, основанный на политике:** Промежуточный вариант. Соседи для пересылки запроса выбираются не случайно, а по правилам: например, запрос шлют тому узлу, который раньше чаще других успешно находил данные или у которого больше всего связей.

33. 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.

Процесс — это программа в момент её выполнения. ОС создает для каждой программы «виртуальный процессор», чтобы они могли работать одновременно.

• **Контент процесса:** Включает **адресное пространство** (код программы, данные и стек в памяти) и **контекст** (значения регистров, список открытых файлов, идентификаторы).

• **Жизненный цикл:** Создание -> Выполнение -> Уничтожение. В процессе работы он может быть «готов», «выполняться», «заблокирован» (ждет данных) или «приостановлен».

• **Системные вызовы:** Это обращения программы к ядру ОС (например, для создания нового процесса). При системном вызове процессор переключается из **пользовательского режима** в **привилегированный режим ядра**.

• **Идентификация:** Каждому процессу присваивается номер **PID**, а также ID пользователя (**UID**) и группы (**GID**), от имени которых он запущен.

• **Дерево процессов:** Процессы могут порождать «дочерние» процессы. Так выстраивается иерархия (дерево), где «прародителем» всех процессов является процесс с **PID=1**.

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

Процессы часто должны работать сообща, обмениваться данными и синхронизироваться. Это и называется **IPC**.

• **Методы взаимодействия:**

◦ **Каналы (pipes):** Связывают вывод одной программы со вводом другой.

◦ **Очереди сообщений:** Список сообщений в памяти ядра, которые процессы могут пересылать друг другу.

◦ **Разделяемая память:** Участок памяти, доступный сразу нескольким процессам (самый быстрый способ).

◦ **Сокеты:** Обеспечивают связь между процессами как на одном ПК, так и по сети.

• **Сигналы:** Программные прерывания. Когда процессу приходит сигнал, он временно прекращает работу, выполняет процедуру обработки сигнала и возвращается в исходное состояние.

• **Семафоры и мьютексы:** Примитивы синхронизации. **Семафор** — это счетчик для управления доступом к ресурсам. **Мьютекс** — это «замок»: только тот поток, который его закрыл (захватил ресурс), может его открыть.

• **Взаимоблокировка (Deadlock):** Ситуация «тупика», когда два процесса ждут друг друга и не могут продолжить работу.

• **Недостатки процессов:** Они «тяжеловесны» (требуют много ресурсов на создание) и изолированы друг от друга, что затрудняет быстрый обмен данными.

35. 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.

Поток — это более мелкая единица дробления внутри процесса. Один процесс может содержать несколько потоков, которые используют общую память этого процесса.

• **Преимущества потоков:** Они создаются в 10–100 раз быстрее процессов. Потоки позволяют приложению не «зависать»: например, в текстовом редакторе один поток может проверять орфографию, а другой — обрабатывать ввод текста.

• **Реализация:**

◦ **На пользовательском уровне:** Управляются библиотекой программы. **Плюсы:** очень быстрое переключение, не нужно обращаться к ядру ОС. **Минус:** если один поток вызовет блокирующую операцию (например, чтение файла), «уснет» весь процесс со всеми остальными потоками.

◦ **В ядре:** Управляются самой ОС. Каждый поток — это независимый объект планирования. Это решает проблему блокировки, но создание таких потоков обходится «дороже».

• **Потоки vs Группы процессов:** Использование процессов безопаснее (у них раздельные пространства данных, защищенные аппаратно), но потоки гораздо эффективнее для быстрой совместной работы с общими данными.

36. 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.

Виртуализация — это создание логической абстракции вычислительных ресурсов, при которой от пользователя скрывается реальная техническая реализация. Простыми словами: пользователь работает с удобным ему представлением объекта (например, целым компьютером), не задумываясь о том, как это устроено внутри физического «железа».

• **Виды интерфейсов в системе:**

1. **ISA (Instruction Set Architecture):** Набор машинных команд, интерфейс между аппаратными средствами и ПО.
2. **Системные вызовы:** Интерфейс операционной системы (библиотеки функций).
3. **API (Application Programming Interface):** Уровень библиотечных вызовов приложений.

• **Принципы виртуализации:** Суть заключается в замене существующих интерфейсов на другие, которые имитируют поведение иных систем. Например, эмулятор может «переводить» системные вызовы Windows в понятные для ОС Linux команды.

• **Способы виртуализации:** По уровню интерфейса выделяют виртуализацию на уровне **набора команд (ISA)**, на уровне **системных вызовов** и на уровне **API**.

• **Виртуализация ЭВМ и ресурсов сервера:** Это размещение нескольких логических машин внутри одной физической. Это позволяет каждому приложению «видеть» только свои ресурсы (процессор, память), полагая, что ему выделен целый сервер.

• **Эффективность и роль в РИС:** Виртуализация отделяет программное обеспечение от оборудования, помещая его в изолированный контейнер. В распределенных системах это ключевая технология, позволяющая создавать облачные вычисления, гибко масштабировать ресурсы и запускать несовместимые приложения на одном хосте.

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

История развития:

• **1960-е:** Появление термина **Hypervisor** (гипервизор) для IBM 360/65. В 1966–67 гг. были созданы первые эмуляторы (CP-40, CP-67).

• **1970-е:** Выход системы **VM/370**, которая поддерживала виртуализацию на десятилетия раньше, чем она стала популярной на ПК.

• **1990–2000-е:** Адаптация технологий для архитектуры Intel x86. В 1999 году появилась **VMware Virtual Platform**. Позже на рынок вышли Microsoft (Hyper-V), Citrix (Xen), Oracle (VirtualBox) и RedHat (KVM).

Требования Попека и Голдберга (1974 г.) к архитектуре ЭВМ: Чтобы компьютер мог эффективно поддерживать виртуализацию, он должен отвечать трем условиям:

1. **Безопасность:** Гипервизор должен иметь полный контроль над всеми ресурсами.
2. **Эквивалентность:** Программа в виртуальной машине должна вести себя точно так же, как на реальном оборудовании.
3. **Эффективность:** Большая часть команд должна выполняться напрямую на процессоре без вмешательства гипервизора.

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

Проблемы архитектуры Intel x86: Долгое время процессоры x86 не соответствовали требованиям виртуализации. Главная проблема — наличие **«чувствительных» команд** (например, POPF), которые меняют состояние процессора, но не вызывают прерывания при выполнении в пользовательском режиме. В результате гипервизор не мог их перехватить и правильно обработать.

Два способа реализации гипервизора (монитора виртуальных машин):

1. **I тип — «Родной» (Native):** Устанавливается прямо на «голое железо» как специализированная ОС (например, VMware ESXi). Он максимально эффективен, но требует собственных драйверов для оборудования.
2. **II тип — Хостинговый (Hosted):** Работает поверх обычной (хозяйской) ОС как обычное приложение (например, VMware Workstation). Проще в установке, но имеет чуть большие накладные расходы.

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

Для решения проблемы команд, которые нельзя напрямую виртуализировать на x86, применяют две стратегии:

1. **Двоичная трансляция (Полная виртуализация):** Гипервизор на лету сканирует код гостевой ОС и заменяет проблемные команды на безопасные аналоги, которые можно виртуализировать. Это обеспечивает полную прозрачность — гостевую ОС не нужно менять.
2. **Паравиртуализация:** Код ядра гостевой ОС заранее модифицируется. В него добавляется специальный набор API (гипервызовы), через которые ОС напрямую общается с гипервизором, исключая использование «опасных» команд. Это повышает производительность, но требует внесения изменений в саму ОС.

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

Выделяют два основных типа ВМ в зависимости от их задач:

- **ВМ процесс:** Создается специально для работы одного конкретного приложения и уничтожается сразу после его завершения. Она создает изолированную среду для выполнения процесса, часто на базе другого набора команд.
  - **Примеры ПО:**
    - **Wine** (для запуска Windows-приложений в Linux).
    - **QEMU** (в режиме динамической трансляции кода, например PowerPC на x86).
    - **HLL VM** (ВМ языков высокого уровня): **Java Virtual Machine (JVM)** и среда выполнения **Python**.
- **ВМ система:** Полноценная виртуальная машина, которая поддерживает работу целой операционной системы со всеми её процессами и приложениями. Примеры: VMware ESXi, Microsoft Hyper-V.

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

Монитор виртуальных машин (**VMM**), или **гипервизор**, — это программный слой, который позволяет нескольким виртуальным машинам одновременно использовать ресурсы одного физического компьютера. Существует три основных варианта его организации:

1. **Native (естественная или полная):** Гипервизор представляет собой тонкий программный уровень, который устанавливается непосредственно на «голое железо» (аппаратное обеспечение) без участия сторонних операционных систем. Главное преимущество такого подхода — максимально высокая **производительность**.
2. **Hosting (хостинг):** В этой схеме виртуальная машина работает не напрямую с оборудованием, а «поверх» уже установленной на компьютер полноценной операционной системы, которую называют **хозяйской (хостовой)**.
3. **Hybrid (гибридная):** Это промежуточный вариант, при котором гипервизор использует оборудование совместно с уже существующей операционной системой.

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

Гипервизор первого типа (также называемый «родным» или *native*) — это специализированная операционная система, которая берет на себя управление всеми командами процессора и распределением ресурсов.

• **Достоинства:**

◦ Высокая эффективность использования ресурсов, так как отсутствует лишняя прослойка в виде «хозяйской» ОС.

◦ **Прозрачность**: не требуется вносить изменения в саму гостевую операционную систему (она «думает», что работает на реальном железе).

• **Недостатки:**

◦ Необходимость иметь собственные драйверы для всех внешних устройств (сетевых карт, контроллеров), что усложняет разработку и поддержку гипервизора.

◦ Меньшая гибкость в использовании специфических функций оборудования.

Решение проблемы работы гостевой ОС в режиме ядра: В архитектуре процессоров x86 есть уровни защиты (кольца). Самое привилегированное — **Кольцо 0** (режим ядра). Гипервизору для управления ресурсами нужно находиться именно там. Проблема в том, что гостевая ОС тоже «хочет» работать в Кольце 0. Чтобы они не конфликтовали, используется **Кольцо 1**: гипервизор занимает Кольцо 0, приложения гостя — Кольцо 3, а саму гостевую ОС помещают в Кольцо 1. Если гостевая ОС пытается выполнить привилегированную команду, процессор вызывает прерывание, и гипервизор сам выполняет эту операцию от имени гостя после проверки.

43. Гипервизор второго типа, достоинства и недостатки.

Этот тип гипервизора работает внутри обычной операционной системы как прикладная программа.

• **Принцип работы:** Он перехватывает «чувствительные» команды гостевой ОС и на лету перезаписывает их безопасным кодом (это называется **двоичной трансляцией**).

• **Достоинства:** Простота использования. Вам не нужно переустанавливать ОС на сервере; гипервизор запускается как обычное окно в Windows или Linux.

• **Недостатки:** Более высокие накладные расходы. На поддержание такой системы уходит **10–15%** мощности процессора.

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

Это технология, при которой гостевая операционная система «знает», что она виртуализирована, и активно помогает гипервизору.

• **Достоинства:** Очень низкие накладные расходы (всего **5–10%**), высокая производительность и возможность создавать среду с разными типами ОС одновременно. Отсутствует потребность в сложной «хозяйской» ОС.

• **Недостатки:** Ядро гостевой ОС **необходимо модифицировать** (изменять код), чтобы встроить в него специальный API для общения с гипервизором. Это невозможно сделать для закрытых систем (например, старых версий Windows), если у разработчика нет доступа к их исходному коду.

Управление памятью: При паравиртуализации гостевая ОС не пытается напрямую управлять физическими таблицами страниц памяти. Вместо этого она взаимодействует с гипервизором через **гостевой API** (гипервызовы VMI). Гипервизор предоставляет гостю готовые механизмы работы с памятью, что исключает конфликты между разными виртуальными машинами.

45. Платформа виртуализации Hyper-V.

Hyper-V — это система аппаратной виртуализации от Microsoft, интегрированная в Windows Server.

Ключевые особенности её архитектуры:

• **Микроядерный подход:** Сам гипервизор Hyper-V очень тонкий и выполняет только базовые задачи — изоляцию разделов и управление памятью. В нём нет драйверов устройств или сложного программного стека.

• **Родительский раздел:** Все драйверы и основной стек виртуализации находятся в специальном разделе ОС, называемом «родительским». Это позволило Microsoft использовать уже существующие драйверы Windows без их переделки под гипервизор.

• **Безопасность и стабильность:** Благодаря тому, что драйверы вынесены из ядра гипервизора, ошибка в одном драйвере не обрушит всю систему виртуализации. Кроме того, малый размер ядра гипервизора сокращает «фронт атаки» для хакеров.

• **Управление:** Для работы в масштабах предприятия используется дополнение — **System Center Virtual Machine Manager** (SCVMM), которое позволяет создавать библиотеки шаблонов и переносить серверы в виртуальную среду (технология P2V).

46. Платформа виртуализации VMware.

VMware является одним из пионеров в области виртуализации, запатентовавшим свои программные техники ещё в 1998 году. Платформа предлагает широкий спектр продуктов: от решений для персональных компьютеров до мощных систем управления центрами обработки данных.

• **Основные продукты:**

◦ **VMware Workstation:** ориентирована на профессиональных пользователей и разработчиков, работая как гипервизор второго типа (поверх обычной ОС).

◦ **VMware ESXi (ранее ESX):** это «родной» гипервизор первого типа, который устанавливается прямо на оборудование сервера без участия сторонней операционной системы,,.

◦ **VMware vSphere:** комплексный продукт для создания надежной виртуальной инфраструктуры ЦОД.

• **Технологическая особенность:** В процессорах x86 есть 17 проблемных инструкций, которые мешают виртуализации. VMware разработала **адаптивную технологию**, которая «перехватывает» эти инструкции в момент выполнения и заменяет их на безопасные аналоги.

• **Управление:** Для масштабируемости используется **VMware vCenter Server**, который служит центральной консолью управления всеми виртуальными средами предприятия.

47. Гипервизор Xen. Особенности.

Xen зародился как исследовательский проект в компьютерной лаборатории Кембриджского университета под руководством Иана Пратта. В 2007 году компания-разработчик XenSource была приобретена Citrix Systems.

Ключевые особенности Xen:

• **Паравиртуализация:** Изначально Xen был самой развитой платформой, использующей этот подход. Это означает, что ядро гостевой операционной системы должно быть **незначительно модифицировано** для работы в виртуальной среде,.

• **Интерфейс VMI:** Гостевая ОС общается с гипервизором через специальные **гипервызовы VMI** (Virtual Machine Interface).

• **Открытость:** Бесплатная версия Xen (Open Source) включена во многие дистрибутивы Linux (Red Hat, SUSE, Debian) и активно используется для образовательных и научных целей,.

• **Коммерческие версии:** Для производственных сред используются продукты **Citrix**, такие как **XenServer** (консолидация серверов) и **XenDesktop** (виртуализация рабочих мест),.

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

Этот метод виртуализирует сервер на уровне самой ОС, создавая изолированные среды, которые называют **контейнерами** (CT) или виртуальными частными серверами (VPS),.

• **Принцип работы:** Все контейнеры используют **одно общее ядро** базовой операционной системы, но имеют собственное сетевое и аппаратное окружение,. Для пользователя контейнер выглядит как полноценный отдельный сервер,.

• **Технологии:** В Linux это реализуется через механизмы **namespaces** (изоляция ресурсов) и **cgroups** (контрольные группы для ограничения потребления ресурсов),,.

• **Достоинства:**

◦ **Легковесность и скорость:** запуск контейнера происходит почти мгновенно, так как не нужно загружать новую ОС,.

◦ **Минимальные накладные расходы:** потери производительности составляют всего около **3%**,.

◦ **Высокая плотность:** на одном сервере можно запустить гораздо больше контейнеров, чем традиционных виртуальных машин.

• **Недостатки:**

◦ **Однородность:** нельзя запустить гостевую ОС с ядром, отличным от базового (например, нельзя запустить Windows в контейнере на Linux),,,.

◦ **Слабая изоляция:** защита контейнеров друг от друга ниже, чем у систем с полноценным гипервизором.

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

Облачные вычисления — это модель предоставления удобного сетевого доступа к общему пулу настраиваемых ресурсов (серверам, сетям, хранилищам), которые можно мгновенно получить с минимальными усилиями по управлению.

Пять обязательных признаков облака (модель NIST):

1. **Самообслуживание по запросу:** пользователь сам получает ресурсы (например, серверное время), не общаясь с персоналом провайдера,.
2. **Широкополосный доступ:** услуги доступны через интернет с любого устройства (ноутбук, смартфон) через стандартные механизмы (браузер),.
3. **Пулы ресурсов:** ресурсы провайдера объединены для обслуживания множества клиентов и динамически перераспределяются между ними,.
4. **Быстрая эластичность:** ресурсы можно мгновенно добавлять или удалять в зависимости от текущих нужд бизнеса,.
5. **Измеряемое обслуживание:** плата взимается только за фактически потребленный объем (как за электричество),.

Облачные вычисления стали результатом **синтеза технологий**: роста мощности процессоров, развития виртуализации, появления быстрых каналов связи и перехода к сервисно-ориентированной архитектуре (SOA),.

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

Облака стали следующим шагом в развитии корпоративных информационных систем (КИС). Если раньше компании строили сложные локальные сети, то теперь архитектура приложений стала **сетецентрической**: данные и вычисления переносятся с персональных компьютеров в «облако» удаленных серверов,.

Типы облаков по модели развертывания:

• **Частное облако (Private cloud):** инфраструктура используется исключительно одной организацией; она может находиться как внутри компании, так и у внешнего провайдера,.

• **Публичное облако (Public cloud):** ресурсы открыты для свободного использования широкой публикой (примеры: Amazon AWS, Microsoft Azure, Google Cloud),,.

• **Гибридное облако (Hybrid cloud):** сочетание двух и более разных облаков (например, часть данных в частном, часть — в публичном), которые связаны технологиями, позволяющими данным «кочевать» между ними,.

• **Облако сообщества (Community cloud):** инфраструктура для группы организаций, имеющих общие задачи (например, единые требования к безопасности или общую миссию),.

51. Препятствия для разработки облачных приложений и сервисов.

Несмотря на популярность облаков, существует ряд серьезных барьеров, которые мешают компаниям полностью перейти на эту модель:

• **Доступность сервиса:** Главный вопрос — что произойдет, если провайдер не сможет предоставить услугу в критический момент. Крупным корпорациям нужны твердые гарантии, что их деятельность не пострадает от перегрузки облака.

• **Привязка к поставщику (Vendor lock-in):** Как только клиент размещает свои данные и приложения у одного провайдера, ему становится крайне трудно и дорого перейти к другому из-за отсутствия единых стандартов.

• **Конфиденциальность и аудит:** Безопасность данных и возможность проверить, кто и когда имел к ним доступ, остаются важнейшими проблемами.

• **Узкие места при передаче данных:** Пересылка огромных объемов информации (терабайтов) через обычный интернет может занимать дни, что делает облако неэффективным для некоторых задач.

• **Непредсказуемость производительности:** Поскольку ресурсы облака общие для многих пользователей, действия одного «соседа» могут случайно замедлить работу приложений другого.

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

На основе анализа рисков и технических ограничений, в облако не рекомендуется переносить:

• **Приложения с гигантскими объемами данных:** Если программе требуется постоянно перекачивать терабайты данных туда и обратно, узкая полоса пропускания сети сделает её работу невыносимо медленной.

• **Системы с жесткими требованиями к задержкам:** Приложения, где важна каждая миллисекунда (например, высокочастотная торговля или управление сложным промышленным оборудованием), могут страдать от непредсказуемых сетевых задержек интернета.

• **Сверхсекретные системы:** Если политика безопасности организации запрещает хранение данных на чужих физических серверах или требует полного физического контроля над оборудованием, публичное облако не подойдет.

• **Приложения, не поддерживающие виртуализацию:** Некоторые старые (унаследованные) программы могут быть жестко привязаны к специфическому «железу», которое невозможно эмулировать в облаке.

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

Масштабируемость — это способность системы справляться с растущей нагрузкой. Методы зависят от того, как устроено приложение:

• **Вертикальное масштабирование:** Это увеличение мощности уже существующей виртуальной машины (добавление ядер процессора или объема ОЗУ). Этот метод часто используется для **SQL-баз данных**, так как их сложно распределить на много компьютеров.

• **Горизонтальное масштабирование:** Это добавление новых виртуальных машин в систему. Это основной метод для облаков.

• **Автомасштабирование (Auto Scaling):** Система сама следит за нагрузкой и автоматически запускает новые копии серверов в часы пик, а когда нагрузка падает — выключает лишние, чтобы сэкономить деньги.

• **Масштабирование NoSQL баз данных:** В отличие от классических БД, системы NoSQL (например, Cosmos DB) изначально созданы для горизонтального роста — они делят данные на сегменты и распределяют их по множеству узлов.

54. Основные модели распределенных вычислений: P2P (одноранговые) вычисления; кластерные вычисления; вычисления по запросу (utility computing); грид-вычисления; облачные вычисления; туманные вычисления (fog computing); гетерогенные вычисления (jungle computing).

Развитие технологий породило несколько способов организации вычислений:

• **P2P (одноранговые):** Модель, где нет главного сервера. Каждый компьютер — одновременно и клиент, и сервер, а участники обмениваются ресурсами напрямую.

• **Кластерные вычисления:** Набор одинаковых компьютеров, соединенных быстрой локальной сетью и работающих как один мощный ресурс под управлением общего узла.

• **Вычисления по запросу (Utility computing):** Модель «вычислений как услуги», где ресурсы предоставляются по мере надобности, а оплата идет только за использованный объем.

• **Грид-вычисления (Grid):** Сеть, объединяющая ресурсы кластеров разных организаций. Это похоже на единую электросеть, где потребитель просто «включается в розетку», не зная, чьи именно мощности он использует.

• **Облачные вычисления:** Перенос задач с личных ПК в «облако» серверов через интернет, где все детали инфраструктуры скрыты от пользователя.

• **Туманные вычисления (Fog computing):** Часть данных обрабатывается прямо на «умных» устройствах (датчиках, маршрутизаторах), а часть — в облаке. Это снижает нагрузку на сеть, что критично для Интернета вещей.

• **Гетерогенные вычисления (Jungle computing):** Сочетание максимально разнообразных и мощных систем для решения сложнейших задач, требующих предельной производительности

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

Предпосылки возникновения:

1. **Аппаратный рост:** Резкое увеличение мощности процессоров, емкости памяти и скорости сетевых каналов.
2. **Зрелость виртуализации:** Появление надежных инструментов (VMware, Hyper-V, KVM), позволяющих делить один сервер на сотни виртуальных.
3. **Экономические факторы:** Кризис 2007 года привел к накоплению лишних мощностей у гигантов вроде Amazon, которые решили начать сдавать их в аренду.
4. **Сервисный подход:** Распространение архитектуры SOA и веб-сервисов (RESTful), позволивших программам легко общаться через интернет.

Ключевые особенности (по модели NIST):

- **Самообслуживание по запросу:** Вы сами заказываете сервер через панель управления без звонков провайдеру.
- **Широкополосный доступ:** Работа через браузер с любого устройства — от смартфона до ноутбука.
- **Объединение ресурсов в пулы:** Провайдер «сваливает» всё железо в общую кучу и динамически распределяет его между клиентами.
- **Быстрая эластичность:** Возможность мгновенно расширить ресурсы в 10 раз и так же быстро их вернуть.
- **Измеряемое обслуживание:** Вы платите только за «нагоревшие» гигабайты и часы, как за электричество.

56. Основные свойства (характеристики) облачных вычислений.

Согласно международным стандартам, облачная система должна обладать пятью ключевыми характеристиками:

1. **Самообслуживание по запросу:** Пользователь может самостоятельно получать вычислительные ресурсы (например, серверное время или место в хранилище) через панель управления, не обращаясь к персоналу провайдера,.
2. **Широкополосный доступ к сети:** Все услуги доступны через интернет с любых устройств: компьютеров, смартфонов, планшетов или ноутбуков,. Для работы пользователю нужен лишь обычный веб-браузер.
3. **Объединение ресурсов в пулы:** Провайдер собирает всё свое оборудование в «общие кучи» (пулы) и динамически распределяет его между множеством клиентов. При этом пользователь обычно не знает точно, на каком именно физическом сервере или в какой стране лежат его данные.
4. **Быстрая эластичность:** Ресурсы можно мгновенно наращивать или уменьшать в зависимости от текущих нужд бизнеса. Система делает это автоматически: если на сайт зашло много людей, она добавит мощностей, если нагрузка упала — отключит лишнее.
5. **Измеряемое обслуживание:** Облако работает как счетчик электроэнергии. Провайдер точно фиксирует объем памяти, время работы процессора и трафик, а клиент платит только за то, что реально использовал,.

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

Определение NIST: Облачные вычисления — это модель предоставления удобного сетевого доступа по запросу к общему пулу настраиваемых ресурсов (серверов, сетей, хранилищ), которые можно мгновенно подготовить и выдать пользователю с минимальными усилиями по управлению.

Модели предоставления услуг:

• **IaaS (Инфраструктура как услуга):** Клиент арендует «чистое железо» в виртуальном виде: процессоры, диски, сети. На эту основу он сам устанавливает операционные системы и нужные ему программы. Примеры: Amazon EC2 и S3.

• **PaaS (Платформа как услуга):** Клиенту предоставляется готовая среда для разработки и тестирования программ,. Программисты могут просто писать код и запускать его, не заботясь о настройке серверов или обновлении ОС. Примеры: Google App Engine, Force.com.

• **SaaS (Программное обеспечение как услуга):** Самый простой уровень, когда пользователь получает готовое приложение через браузер,. Не нужно покупать лицензии или устанавливать софт на компьютер. Примеры: Gmail, Google Docs, Salesforce,.

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

Облака различаются по тому, кто имеет к ним доступ:

• **Частное облако (Private cloud):** Создается для использования только одной конкретной организацией. Оно может находиться как на территории самой компании, так и у стороннего провайдера, но ресурсы ни с кем не делятся,.

• **Публичное облако (Public cloud):** Инфраструктура открыта для всех желающих — от обычных людей до крупных корпораций. Владеет таким облаком провайдер (например, Amazon, Google или Microsoft),.

• **Облако сообщества (Community cloud):** Группа организаций с общими интересами (например, банки или государственные органы) объединяют свои ресурсы для решения совместных задач и соблюдения единых правил безопасности,.

• **Гибридное облако (Hybrid cloud):** Это сочетание двух и более типов облаков (например, частного и публичного), которые остаются отдельными объектами, но связаны технологиями для обмена данными и приложениями,.

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

Проблемы с SLA (Соглашение об уровне обслуживания):

• Клиентам сложно оценить реальные гарантии провайдеров, так как многие из них пишут SLA таким образом, чтобы максимально защитить себя от исков, предлагая пользователям лишь минимум обязательств.

• Существует риск, что в критической ситуации провайдер может ограничить или вовсе закрыть приложения клиентов с низким приоритетом обслуживания.

Проблемы управления данными:

• Данные в облаке часто огромны, не структурированы и распределены между разными узлами.

• Поскольку физический контроль над серверами находится у провайдера, клиент вынужден полностью полагаться на него в вопросах конфиденциальности (защиты от подглядывания) и аудита (проверки того, что настройки безопасности не менялись),.

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

Безопасность:

• Облака — это лакомый кусочек для хакеров. Основные угрозы включают создание ботнетов (сетей «зомби-компьютеров»), фишинг и кражу данных при их передаче по сети,.

• Системы уязвимы для DDoS-атак, программных ошибок (как это случалось с Amazon ELB) и даже природных явлений, таких как молнии, которые могут вывести из строя целые регионы ЦОД,,.

Совместимость (Интероперабельность):

• Многие облачные сети спроектированы как «закрытые клубы», которые не умеют общаться друг с другом.

• Это создает проблему «привязки к поставщику»: если вы захотите перенести данные от одного провайдера к другому, это может оказаться технически сложным и дорогим удовольствием из-за отсутствия единых стандартов,.

Управление энергетическими ресурсами:

• Огромные центры обработки данных потребляют колоссальное количество энергии: около 53% расходов уходит на электричество и охлаждение.

• Провайдеры ищут компромисс между экономией (отключением неиспользуемых серверов, замедлением процессоров) и сохранением высокой производительности для клиентов,

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

Мультиарендность (мультитенентность) — это ситуация, когда ресурсы одного физического сервера (процессор, память, диски) одновременно используют несколько разных клиентов («арендаторов»).

• **Проблема производительности:** Когда много клиентов обращаются к одному и тому же оборудованию или базам данных, действия одного «соседа» могут замедлить работу других, увеличивая время отклика.

• **Проблема безопасности:** На уровне приложений ресурсы разделяются очень тесно, что создает риски утечки данных между клиентами, если механизмы изоляции сработают некорректно.

Надежность и доступность:

• **Сетевые условия:** В модели SaaS (программное обеспечение как услуга) приложение должно быть доступно пользователю даже при плохом интернете. Например, сбой в сервисе Apple MobileMe показал, что без качественной синхронизации пользователи просто теряют доступ к своим данным.

• **Техногенные и природные факторы:** Облака уязвимы для сбоев питания, программных ошибок (как это было с балансировщиком Amazon ELB) и даже ударов молний, которые могут вывести из строя целые регионы дата-центров.

• **Динамическая нестабильность:** Из-за того, что в облаке взаимодействуют сервисы разных поставщиков (сети, хранилища, приложения), их разные политики управления могут привести к непредсказуемым системным сбоям.

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

Сегодня в облачной индустрии наблюдается дефицит единых правил игры, что мешает системам разных производителей «понимать» друг друга.

• **Привязка к поставщику (Vendor lock-in):** Из-за отсутствия стандартов клиенту, начавшему работать с одним провайдером, очень сложно и дорого перенести свои данные и программы к другому.

• **Отсутствие интеграции:** Многие облачные сети спроектированы как закрытые системы, что мешает компаниям объединять свои ИТ-ресурсы для экономии средств.

Организации, создающие стандарты:

• **NIST:** Опубликовал общепризнанное определение облачных вычислений.

• **DMTF и SNIA:** Эти отраслевые группы определяют стандартные интерфейсы для управления облаком и хранения данных.

• **OASIS:** Продвигает проекты стандартов для структурированной информации.

• **OpenStack Foundation и Open Group:** Фокусируются на открытых стандартах облачных сервисов.

• **TM Forum и Cloud Service Customer Council:** Помогают пользователям правильно составлять соглашения об уровне обслуживания (SLA).

63. Уязвимости облачных систем.

Облака привлекательны для злоумышленников из-за огромной концентрации данных в одном месте.

• **Сетевые угрозы:** Ботнеты (сети зараженных компьютеров-зомби), фишинг и перехват данных при их передаче по сети.

• **DDoS-атаки:** Попытки «затопить» сервер фальшивым трафиком, чтобы он перестал отвечать реальным пользователям. Существуют сложные виды атак, например, через XML-теги, которые истощают ресурсы процессора.

• **Уязвимости виртуализации:**

◦ **Побег из ВМ (VM Escape):** Ситуация, когда вирус из виртуальной машины «вырывается» на уровень гипервизора и получает контроль над всем сервером и другими ВМ.

◦ **Кража ВМ:** Злоумышленник может просто скопировать файл образа виртуальной машины вместе со всеми секретными данными клиента.

◦ **Гиперджекинг (Hyperjacking):** Установка фальшивого гипервизора (руткита), который захватывает управление физическим сервером еще до загрузки основной системы защиты.

64. Препятствия для развития облачных систем.

Существует несколько критических факторов, которые заставляют бизнес осторожничать с облаками:

1. **Доступность сервиса:** Крупные компании боятся переносить важные процессы в облако, не имея 100% гарантии, что провайдер не «упадет» в ответственный момент.
2. **Конфиденциальность:** Передавая контроль над данными сторонней организации, клиент теряет возможность лично проверять настройки безопасности оборудования.
3. **Узкие места в сетях:** Если приложению нужно передать 1 Терабайт данных через медленный интернет (1 Мбит/с), это займет 10 дней. В таких случаях иногда быстрее отправить жесткий диск курьером.
4. **Непредсказуемость:** Из-за совместного использования ресурсов несколькими компаниями скорость работы приложений может внезапно падать.

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

Виртуализация — это создание «цифровой копии» железного компонента, которая работает так же, как настоящая, но на самом деле является программой.

• **Процессор (CPU):** Специальная программа (гипервизор) заставляет гостевую операционную систему верить, что она единственная владеет процессором, хотя на самом деле его мощности делятся между несколькими системами.

• **Память:** Система создает двойной перевод адресов. Сначала из виртуальной памяти программы в «физическую» память виртуальной машины, а затем — в реальную оперативную память физического сервера.

• **Устройства ввода/вывода:** Виртуализируются сетевые карты, USB-порты, дисководы, видеоадаптеры и даже мышь с клавиатурой. Физические диски (HDD/SSD) представляются виртуальной машине как файлы образов.

• **Сеть:** Позволяет объединить ресурсы связи, разделяя общую полосу пропускания на независимые каналы.

◦ **VLAN:** Создает изолированные логические сети внутри одной физической.

◦ **VIP (виртуальный IP):** Позволяет нескольким серверам иметь один общий адрес для балансировки нагрузки.

◦ **VPN:** Обеспечивает защищенный «туннель» для связи через открытый интернет.

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

Виртуализация хранилищ данных — это способ представления ресурсов хранения в абстрактном (логическом) виде, который не привязан к конкретным физическим дискам. Для пользователя или приложения это выглядит как единое удобное пространство (том), хотя на самом деле данные могут быть разбросаны по разным устройствам, подключенным по разным протоколам.

Архитектура LVM (Logical Volume Manager): Это технология управления дисковым пространством, которая строится «снизу вверх»:

1. **Физический том:** В основе лежит реальное блочное устройство (целый диск или его раздел).
2. **Группа томов:** Несколько физических томов объединяются в одну большую «корзину» — единое пространство.
3. **Логический том:** Это пространство «корзины» нарезается на виртуальные разделы (логические тома), которые и видят приложения и файловые системы. Для кластеров существует расширение **CLVM**, которое позволяет нескольким серверам одновременно управлять общим пространством хранения (например, в сетях SAN).

Классификация виртуальных хранилищ включает 5 типов:

- **Блочная** (на уровне блоков данных).
- **Дисковая** (эмуляция физических свойств диска прошивкой).
- **Ленточная** (виртуализация носителей и накопителей для библиотек).
- **Файловая система**.
- **Виртуализация файлов** (автоматический перенос данных между носителями).

67. Блочные виртуальные хранилища данных. Достоинства и недостатки.

В таких системах используются **«сырые» (raw)** тома, которые не имеют заранее заданной структуры. Каждый такой блок операционная система воспринимает как отдельный жесткий диск.

Достоинства:

• **Высокая производительность:** Блочные системы работают быстрее, чем файловые.

• **Гибкость:** Администратор может отформатировать такой «виртуальный диск» под любую файловую систему или установить на него любую ОС.

• **Эффективность:** Использование крупных блоков (например, 4 КБ вместо стандартных 512 байт) позволяет быстрее отслеживать данные и требует меньше служебных записей.

• **Надежность:** Широко используется в сетях хранения данных (SAN), обеспечивая очень эффективную передачу данных.

Недостатки:

• **Ограниченность:** Объем дискового пространства виртуального тома всё же ограничен физическими ресурсами.

• **Длительность процессов:** Операции по миграции (переносу) данных между хранилищами занимают много времени.

68. Файловые виртуальные хранилища данных. Достоинства и недостатки.

Это наиболее распространенный тип систем (например, протоколы NFS или CIFS), где данные хранятся и передаются в виде готовых файлов.

Достоинства:

• **Простота:** Такую систему очень легко реализовать и начать использовать.

• **Совместный доступ:** Пользователи с разными ОС (Windows, Linux, MacOS) могут легко обмениваться файлами.

Недостатки:

• **Отсутствие гарантий:** Сложно обеспечить стабильное качество обслуживания для критически важных задач, так как среда общая для всех пользователей.

• **Уязвимость:** Общее хранилище легче подвергается атакам или сбоям.

• **Нагрузка на сеть:** Интенсивное использование файлового хранилища может сильно «забить» локальную сеть, мешая другим пользователям.

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

Существует три основных метода организации сетевого хранения:

1. **DAS (Direct Attached Storage):** Традиционный метод, когда диски подключены напрямую к физическому серверу. Он прост, но его **сложно виртуализировать**, он плохо масштабируется из-за ограниченного числа портов и часто ресурсы используются неэффективно.
2. **NAS (Network Area Storage):** Это отдельный файловый сервер в сети. Он обеспечивает **единый источник данных**, упрощает создание резервных копий и снижает затраты на администрирование. Время отклика у него быстрое, но всё же медленнее, чем у локального диска.
3. **SAN (Storage Area Network):** Это специализированная высокоскоростная сеть (<10 Гбит), объединяющая мощные системы хранения данных (СХД). Она превращает обычные диски в **сверхнадежные массивы**, доступные круглосуточно, и обладает развитыми средствами управления.

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

Это метод объединения сетевых ресурсов путем разделения доступной полосы пропускания на независимые каналы. Каждый канал может быть назначен конкретному серверу или устройству в реальном времени.

Достоинства:

• **Экономия:** Снижаются затраты на физическое оборудование и электричество.

• **Живучесть:** Позволяет быстро восстановить систему после аварии, автоматически переключая задачи с упавшего хоста на рабочий.

Недостатки:

• **Сложность:** Виртуальная сеть — это сложная структура, которая может привести к снижению производительности.

• **Требования к кадрам:** Администраторы должны быть очень высококвалифицированными, а планирование должно быть максимально продуманным.

Основные технологии:

• **VLAN (Виртуальная локальная сеть):** Создание независимых логических сетей поверх одной физической для изоляции трафика (например, отдельная сеть для управления, отдельная для данных).

• **VIP (Виртуальный IP):** Адрес, не привязанный к конкретной железке. Используется для балансировки нагрузки: если один сервер откажет, VIP-адрес останется доступным, так как на него ответит другой сервер.

• **VPN (Виртуальная частная сеть):** Создает защищенный «туннель» через интернет, позволяя удаленному сотруднику работать так, будто его компьютер физически подключен к сети компании.

71. Виртуализация десктопов. Преимущества и недостатки.

Виртуализация десктопа — это технология, которая отделяет рабочую среду пользователя (операционную систему, приложения и данные) от его физического компьютера. В этой модели всё «содержимое» вашего компьютера живет на удаленном сервере в центре обработки данных, а на монитор пользователя передается только «картинка».

• **Как это работает:** На серверах компании запускается множество виртуальных машин с обычными ОС (Windows 7, 10 или Linux). Пользователь подключается к своей личной виртуальной машине через специальное устройство — **тонкий клиент** или обычный старый ноутбук.

• **Преимущества:**

◦ **Доступ отовсюду:** Можно войти в свой рабочий кабинет из дома, из офиса или в дороге через интернет.

◦ **Безопасность:** Все данные хранятся в защищенном центре обработки данных, а не на диске ноутбука, который можно потерять.

◦ **Простота обновления:** Системному администратору не нужно обходить 100 компьютеров, чтобы обновить программу — он делает это один раз на сервере,.

• **Недостатки:**

◦ **Зависимость от сети:** Если интернет пропадет или будет очень медленным, работать станет невозможно.

◦ **Стоимость:** Тонкие клиенты и мощные серверы для них иногда обходятся так же дорого, как и обычные ПК.

◦ **Ограниченность ОС:** Платформы виртуализации поддерживают не все виды операционных систем.

72. Виртуализация приложений. Достоинства и недостатки.

Это технология, которая позволяет запускать программу без её установки на компьютер в привычном понимании. Программа работает в «изолированном пузыре», который содержит все необходимые ей файлы и настройки реестра.

• **Как это работает:** Приложение упаковывается в специальный контейнер вместе со своими библиотеками и настройками. Этот контейнер доставляется на компьютер пользователя через интернет или USB-флешку и запускается в виртуальной среде, не внося никаких изменений в основную ОС,.

• **Достоинства:**

◦ **Нет конфликтов:** Можно одновременно запустить две разные версии одной и той же программы, которые обычно конфликтуют (например, разные версии Word).

◦ **Безопасность:** Приложение изолировано от системы, что снижает риск заражения компьютера.

◦ **Быстрота:** Не нужно ждать долгой установки — приложение готово к работе сразу после загрузки контейнера.

• **Недостатки:**

◦ **Требования к сети:** Для доставки приложений пользователям нужна стабильная и быстрая связь.

73. Микросервисная архитектура в облачных приложениях.

Микросервисная архитектура — это современный подход к созданию программ, когда одно большое приложение разбивается на множество мелких, независимых частей (**микросервисов**),.

• **Суть:** Каждый микросервис выполняет только одну конкретную задачу (например, один отвечает за корзину в интернет-магазине, другой — за оплату, третий — за поиск товаров).

• **В облаке:** Облачные провайдеры предлагают специальные инструменты для работы с такой архитектурой. Например, **AWS Step Functions** помогает визуализировать и координировать работу этих частей, а **AWS X-Ray** позволяет отслеживать запросы между тысячами микросервисов, чтобы быстро найти причину ошибки,.

74. Бессерверная технология реализации облачных приложений и служб.

Это способ запуска приложений, при котором разработчику вообще **не нужно думать о серверах**, их настройке, обновлении или масштабировании.

• **Почему она так называется:** Серверы, конечно, существуют, но ими полностью управляет облачный провайдер (например, Amazon или Microsoft). Разработчик просто загружает свой код, а облако само решает, когда и на какой мощности его запустить.

• **Особенности:**

◦ **Оплата за результат:** Вы платите только за те миллисекунды, когда ваш код реально работал,. Если приложением никто не пользуется, вы платите **0 рублей**.

◦ **Автомасштабирование:** Если на сайт внезапно зайдет миллион человек, облако само выделит нужные мощности.

◦ **Примеры:** Службы **AWS Lambda** или **Azure Functions**,.

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

Облачная архитектура строится как **внешний интерфейс (API)**, который дает доступ к огромному общему пулу ресурсов (процессорам, памяти, дискам),.

Основные проблемы дизайна таких систем:

1. **Привязка к поставщику (Vendor lock-in):** Если вы построили систему под Amazon, перенести её на Microsoft Azure будет очень сложно и дорого, так как стандарты у всех разные,.
2. **Доступность и надежность:** Что делать, если у провайдера случится сбой (например, из-за удара молнии в дата-центр)? Крупные компании боятся доверять облаку свои самые важные процессы,.
3. **Безопасность и конфиденциальность:** Ваши данные лежат на чужих серверах. Провайдер должен гарантировать, что никто другой (включая сотрудников провайдера) их не увидит,.
4. **Управление ресурсами (SLA):** Провайдеры часто обещают «золотые горы», но на деле составляют договоры так, чтобы нести минимум ответственности за простои.
5. **Энергопотребление:** Огромные облака потребляют колоссальное количество энергии на работу и охлаждение (около 53% всех расходов дата-центра), что заставляет инженеров постоянно искать способы экономии.

76. Архитектурные принципы и свойства веб сервисов REST.

REST (Representational State Transfer) — это современный архитектурный стиль, который является частью сервисного подхода к созданию информационных систем.

• **Ресурсо-центрированность:** В отличие от традиционных веб-сервисов, REST предлагает рассматривать распределенную систему как **коллекцию ресурсов**. Каждый ресурс индивидуально управляется своим компонентом (сервисом).

• **Использование стандартов Web:** RESTful-сервисы полностью интегрированы в среду Web и используют стандартные протоколы (преимущественно **HTTP**) для взаимодействия.

• **Идентификация:** Каждый ресурс должен быть четко идентифицирован, обычно с помощью **URI** (уникального адреса).

• **Универсальный интерфейс:** Для доступа к ресурсам используются стандартные команды. Например, облачные хранилища (такие как Amazon S3) предоставляют **RESTful API**, где операции размещения или получения данных выполняются с помощью стандартных HTTP-запросов.

• **Масштабируемость:** Этот стиль позволяет строить высокомасштабируемые системы за счет простоты взаимодействия компонентов.

77. Контейнеры. Контейнерная виртуализация в Linux.

Контейнеризация — это метод виртуализации, который работает на уровне операционной системы, а не оборудования.

• **Принцип работы:** В отличие от обычных виртуальных машин, контейнеры используют **одно общее ядро** базовой операционной системы. Для пользователя контейнер выглядит как полноценный отдельный сервер (VPS), но внутри он разделяет ресурсы ядра с другими контейнерами.

• **Ключевые технологии в Linux:**

◦ **namespaces (пространства имен):** механизм, который изолирует процессы друг от друга, создавая для них иллюзию работы на отдельной машине (изоляция сети, пользователей, идентификаторов процессов PID и т.д.).

◦ **cgroups (контрольные группы):** отвечают за ограничение ресурсов. Они следят, чтобы контейнер не потреблял больше памяти или процессорного времени, чем ему положено.

◦ **chroot:** старейший механизм «изменения корня», который позволяет запереть программу в отдельной папке, делая остальную файловую систему недоступной для неё.

• **Популярные решения: LXC** (базовая технология), **Docker** (самая известная надстройка над LXC), **OpenVZ** и **rkt**.

• **Преимущества:** Минимальные потери производительности (всего около **3%**), очень быстрый запуск и возможность запускать в десятки раз больше окружений на одном сервере, чем при обычной виртуализации.

78. Контейнеры Windows Server и Hyper-V.

Microsoft также внедрила технологии контейнеризации в свою экосистему, адаптировав их под особенности Windows.

• **HCSShim:** Для управления контейнерами была создана специальная системная оболочка совместимости (**HCSShim**).

• **Управление данными:** Система использует специальные драйвер-фильтры хранилища Windows для эффективной работы с образами и файлами контейнеров.

• **Изоляция:** Microsoft предлагает разные уровни изоляции, включая использование технологий гипервизора **Hyper-V** для запуска контейнеров в максимально защищенной среде (хотя они остаются более легковесными, чем полные ВМ).

• **История:** Первые шаги в этом направлении начались еще с продукта **Virtuozzo для Windows** в 2005 году, а современные решения стали частью Windows Server.

79. Использование контейнеров в Microsoft Azure.

Облако Azure предлагает целый набор инструментов для работы с контейнерами (категория «Контейнеры» включает 7 основных служб):

• **Azure Kubernetes Service (AKS):** Самая популярная служба для управления большими кластерами контейнеров.

• **Azure Container Instances (ACI):** Позволяет запускать контейнеры мгновенно, не настраивая виртуальные машины или серверы (модель «контейнер как услуга»).

• **Web App for Containers:** Возможность развертывать веб-приложения напрямую из контейнеров Docker в службе приложений Azure.

• **Azure Service Fabric:** Платформа, которая позволяет создавать и координировать микросервисы, упакованные в контейнеры, обеспечивая их высокую доступность.

• **Поддержка оркестраторов:** Помимо Kubernetes, Azure поддерживает и другие системы управления — **Docker Swarm** и **DC/OS**.

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

Облачное хранилище — это модель, где данные лежат на множестве распределенных в сети серверов, которые для клиента выглядят как один большой виртуальный диск.

- **Основные характеристики:**
  - **Мультитенантность (многоарендность):** Одним физическим хранилищем одновременно пользуются тысячи разных клиентов, надежно изолированных друг от друга.
  - **Масштабируемость:** Можно мгновенно увеличить объем хранилища с гигабайтов до петабайтов.
  - **Управляемость:** Система сама находит ошибки и исправляет их, автоматически подключая новые диски.
  - **Методы доступа:** Доступ через API (REST), сетевые папки (NFS, CIFS) или блочные протоколы (iSCSI).
- **Преимущества:**
  - **Экономия:** Платите только за занятое место, а не за весь сервер.
  - **Доступность:** Данные доступны из любой точки мира с любого устройства (ПК, смартфон).
  - **Надежность:** Провайдер сам делает резервные копии и гарантирует сохранность данных (до 99.999999999%).
- **Недостатки:**
  - **Зависимость от связи:** Если интернет пропал, доступа к файлам нет.
  - **Безопасность:** Существуют риски перехвата данных, если сервис не использует надежное шифрование.
- **Типы хранилищ:** Блочные (как жесткий диск), файловые (как папки), объектные (файлы с метаданными и URL-ссылками) и СУБД (базы данных).

81. Общая архитектура хранения данных в облаке.

Облачная архитектура хранения данных — это прежде всего доставка ресурсов хранения по требованию в высокомасштабируемой и многопользовательской (мультиарендной) среде.

• **Виртуальный сервер:** С точки зрения клиента, облако выглядит как один большой виртуальный сервер, хотя физически данные распределены по многочисленным серверам в сети.

• **Скрытая структура:** Внутренняя структура серверов, их количество и расположение обычно не видны пользователю.

• **Внешний интерфейс:** Обобщенно архитектура представляет собой внешний интерфейс, который предоставляет **API (программный интерфейс)** для доступа к физическим накопителям.

• **Управляемость:** Система должна быть в значительной степени самоуправляемой: она должна уметь автоматически подключать новые диски и самостоятельно находить и исправлять ошибки.

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

Главное отличие облачных систем от традиционных заключается в способах доступа к ним.

Методы доступа:

1. **API веб-сервисов:** Наиболее современный способ, включая **RESTful API** (используется в Amazon S3, Azure).
2. **Файловые протоколы:** Позволяют работать с облаком как с обычной сетевой папкой через протоколы **NFS, CIFS (Samba), FTP** или **WebDAV**.
3. **Блочный доступ:** Доступ на низком уровне, как к физическому жесткому диску, через протокол **iSCSI**.

Облачные модели хранения (по типу развертывания):

- **Частное облако:** Используется внутри одной организации (решения от IBM, Cleversafe).
- **Публичное облако:** Доступно всем желающим через интернет (Amazon, Nirvanix).
- **Гибридное облако:** Сочетание частных и публичных ресурсов.

Типы облачных ресурсов (по способу организации данных):

- **Диск (блочное):** Можно управлять файловой системой, разбивать на разделы; пространство ограничено размером диска.
- **Папка (файловое):** Нельзя менять структуру диска, размер ограничен тарифным планом.
- **Набор данных (СУБД):** Работа с атомарными данными (числа, строки) через базы данных.
- **Объект (объектное):** Данные хранятся вместе с метаданными (описанием свойств), доступ к ним идет через URL-ссылки.

83. Обеспечение производительности облачных хранилищ данных. Протокол FASP.

Главная проблема производительности при передаче больших объемов данных в облако кроется в протоколе **TCP**.

• **Проблема TCP:** Он управляет потоком на основе подтверждений приема; если пакет задерживается, скорость всей передачи падает, что критично при больших расстояниях.

• **Решение — протокол FASP:** Разработан компанией Aspera (используется Amazon) для ускорения массового перемещения данных.

• **Основа на UDP:** В отличие от TCP, FASP базируется на протоколе **UDP**, который позволяет приложению самому управлять скоростью и заторами.

• **Эффективность:** FASP максимально использует доступную полосу пропускания, исключая «узкие места» стандартных сетевых схем.

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

Лекции выделяют ключевые особенности популярных сервисов, которые часто ошибочно называют «облачными дисками», хотя технически это «облачные папки».

• **Google Drive:** Делит общее пространство с Gmail и Google Photo; поддерживает более 30 типов файлов; максимальный объем одного файла — до 5 ТБ.

• **OneDrive (Microsoft):** Интегрирован в Office 365 и Windows; обеспечивает удобный файловый обмен.

• **Dropbox:** Обладает лучшей в классе технологией синхронизации и 256-битным шифрованием; **недостаток** — приложение следит только за одной конкретной папкой Dropbox, а также были заявления о проблемах с конфиденциальностью данных.

• **Яндекс.Диск:** Работает на основе синхронизации; интегрируется в MS Office; имеет функцию автозагрузки фото и видео с внешних носителей.

• **Mega:** Шифрует весь контент прямо в браузере (алгоритм AES); ключи передаются по схеме Friend-to-Friend; считается одним из самых выгодных и защищенных сервисов.

85. Технологии работы с большими данными в облаке. Apache Hadoop. Распределенная файловая система HDFS. Механизм репликации HDFS. Apache Spark - альтернатива Hadoop.

Для обработки гигантских массивов данных, которые не «влезают» в обычные системы, используются кластерные технологии.

Apache Hadoop: Это фундаментальный проект для обработки данных, основанный на парадигме **MapReduce** (разделение задачи на тысячи мелких одинаковых частей).

• **HDFS (Распределенная файловая система):** В ней файлы разбиты на блоки (по 64 или 128 МБ) и разбросаны по узлам кластера.

• **Узлы HDFS:** Сервер имен (**NameNode**) хранит «карту» данных (метаданные), а серверы данных (**DataNode**) хранят сами блоки.

• **Механизм репликации:** Каждый блок данных копируется несколько раз; если один сервер (DataNode) ломается, система автоматически создает новые копии на других узлах, чтобы данные не пропали.

Apache Spark: Современный движок для распределенной обработки, который часто становится популярнее Hadoop.

• **Скорость:** Spark может работать до **100 раз быстрее** Hadoop в оперативной памяти (RAM) за счет эффективных алгоритмов.

• **Альтернатива:** Spark обычно не заменяет весь Hadoop, а заменяет только его модуль вычислений (**MapReduce**), продолжая использовать систему хранения **HDFS**.

• **Применение:** Идеален для машинного обучения и задач, требующих многократных обращений к одному набору данных.

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

Для обработки гигантских объемов информации, которые невозможно обработать на одном компьютере, применяются распределенные технологии, такие как **Apache Hadoop**. Эта система основана на вычислительной парадигме **MapReduce**, суть которой заключается в разделении одной большой задачи на множество мелких одинаковых заданий, выполняемых параллельно на разных узлах кластера.

Принцип работы MapReduce состоит из двух основных этапов:

1. **Этап Map (Отображение):** Главный узел (master node) принимает входные данные, разбивает их на части и передает рабочим узлам (worker nodes) для первичной обработки.
2. **Этап Reduce (Свертка):** Главный узел собирает промежуточные ответы от рабочих узлов и формирует из них окончательный единый результат.

Такой подход позволяет значительно ускорить вычисления за счет того, что данные обрабатываются одновременно на десятках или сотнях серверов. Помимо самого механизма вычислений, Hadoop включает в себя распределенную файловую систему **HDFS** для хранения данных и систему **YARN** для управления ресурсами кластера.

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

Google App Engine (GAE) — это облачная платформа уровня **PaaS** (платформа как услуга), созданная Google для хостинга и разработки масштабируемых веб-приложений,.

Инфраструктура GAE включает ключевые компоненты:

• **GFS (Google File System):** Собственная распределенная файловая система Google.

• **MapReduce:** Сервис для распределенных вычислений.

• **BigTable:** Высокопроизводительная база данных, на которой строятся многие сервисы Google.

• **Chubby:** Сервис распределенной блокировки и онлайн-словарь.

Функциональная модель GAE состоит из следующих элементов:

• **Datastore:** Объектно-ориентированное хранилище структурированных данных, работающее на базе BigTable.

• **Среда исполнения:** Поддерживает разработку на языках **Java** и **Python**.

• **SDK:** Набор инструментов, позволяющий программисту тестировать приложение на своем компьютере перед загрузкой в «облако».

• **Административная консоль:** Удобный интерфейс для управления жизненным циклом приложений.

Преимущества GAE в легкости разработки и отсутствии необходимости обслуживать серверы, однако платформа накладывает ограничения на использование некоторых стандартных библиотек и не поддерживает классические реляционные базы данных.

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

AWS — это мощная инфраструктура облачных веб-сервисов, запущенная компанией Amazon в 2006 году. Она предоставляет вычислительные мощности, хранилища и базы данных с оплатой только за фактически потребленные ресурсы.

Базовые сервисы в архитектуре AWS:

• **EC2 (Elastic Compute Cloud):** Аренда виртуальных серверов,.

• **S3 (Simple Storage Service):** Надежное объектное хранилище для любых объемов данных,.

• **EBS (Elastic Block Store):** Виртуальные диски (блоки) для серверов EC2, которые сохраняются даже после удаления самого сервера,.

• **SQS (Simple Queue Service):** Служба очередей сообщений для связи разных частей приложения,.

• **SNS (Simple Notification Service):** Сервис для рассылки уведомлений пользователям или программам,.

Географическая структура: AWS разделена на **регионы** (географические области по всему миру) и **зоны доступности (AZ)**.

• **Зона доступности** — это один или несколько изолированных дата-центров внутри региона с собственным питанием и сетью.

• Все зоны в одном регионе соединены сверхскоростными каналами связи, что позволяет копировать данные между ними почти мгновенно для защиты от сбоев.

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

Сервис «Вычисления» (Compute):

1. **Amazon EC2:** Основной сервис для запуска виртуальных машин Linux или Windows.
2. **AWS Lambda:** Бессерверные вычисления — вы просто загружаете код, а Amazon сам запускает его в ответ на события (оплата идет только за время работы кода),.
3. **AWS Elastic Beanstalk:** Автоматизирует развертывание веб-приложений: сам настраивает серверы, балансировку нагрузки и масштабирование.
4. **Amazon ECS и ECR:** Инструменты для работы с контейнерами Docker и их образами,.
5. **Auto Scaling:** Автоматически добавляет или удаляет серверы в зависимости от нагрузки на сайт.

Сервис «Хранилище» (Storage):

1. **Amazon S3:** «Бесконечное» хранилище файлов с доступом через интернет,.
2. **Amazon EBS:** Быстрые диски для виртуальных машин EC2 (как жесткий диск в вашем ПК),.
3. **Amazon EFS:** Общая файловая система, которую можно подключить одновременно к многим серверам.
4. **Amazon Glacier:** Очень дешевое хранилище для архивов, которые редко нужны (например, старые отчеты или бэкапы).
5. **AWS Storage Gateway:** Позволяет соединить ваше офисное оборудование с облачным хранилищем Amazon.

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

Сервис «База данных» (Databases):

1. **Amazon RDS:** Управляемые классические базы данных (MySQL, PostgreSQL, Oracle, SQL Server).
2. **Amazon Aurora:** Сверхбыстрая база данных от Amazon, совместимая с MySQL и PostgreSQL, но работающая в 5 раз быстрее.
3. **Amazon DynamoDB:** Быстрая NoSQL база данных для огромных нагрузок с задержкой менее 10 миллисекунд.
4. **Amazon ElastiCache:** Сервис для ускорения приложений за счет хранения данных в оперативной памяти (Redis, Memcached).
5. **Amazon Redshift:** Хранилище для анализа огромных массивов данных (петабайтного масштаба).
6. **Amazon Neptune:** Специальная база данных для работы со сложными связями (графами).

Сервис «Миграция» (Migration): Предназначен для переноса данных и серверов из вашего офиса в облако AWS.

1. **AWS Database Migration Service (DMS):** Позволяет переносить базы данных в облако практически без остановки работы приложения.
2. **AWS Server Migration Service (SMS):** Автоматизирует перенос тысяч реальных серверов в виртуальные машины AWS.
3. **Семейство Snowball:** Физические защищенные устройства (чемоданы с дисками) для перевозки огромных объемов данных (ТБ и ПБ), если интернет слишком медленный,.
4. **AWS Snowmobile:** Целый грузовик-контейнер для перевозки эксабайтов данных (до 100 ПБ в одном контейнере).

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

Amazon Web Services (AWS) предлагает широкий выбор инструментов для хранения информации, каждый из которых подходит для своих задач.

• **Основные типы хранилищ:**

◦ **Amazon S3:** Универсальное объектное хранилище для любых объемов данных, которое часто называют «бесконечным».

◦ **Amazon EBS:** Виртуальные диски для серверов, которые не исчезают после выключения виртуальной машины.

◦ **Amazon EFS:** Общая файловая система, которую могут одновременно использовать несколько серверов.

◦ **Amazon Glacier:** Очень дешевое место для хранения архивов, которые редко требуются пользователю.

Apache Hadoop в Amazon EMR: Для обработки гигантских массивов данных AWS предоставляет сервис **Amazon EMR**. Это управляемая платформа, которая позволяет быстро запускать кластеры **Hadoop**, **Spark** и другие системы распределенных вычислений. Особенность EMR в том, что она использует специальную файловую систему **EMRFS**, которая позволяет Hadoop работать напрямую с данными, хранящимися в **Amazon S3**. Это делает обработку данных дешевле и проще, так как вам не нужно постоянно держать запущенным огромный кластер серверов.

92. Платформа Microsoft Microsoft Azure. Обзор служб Microsoft Azure.

Microsoft Azure — это открытая и гибкая облачная платформа, которая позволяет создавать и запускать приложения через глобальную сеть дата-центров Microsoft.

• **Основные особенности:** Она поддерживает любые языки программирования и инструменты разработки. Трафик между дата-центрами Azure проходит по собственной закрытой сети Microsoft, не выходя в обычный интернет, что повышает безопасность.

• **Категории служб:** Azure включает более **100 служб**, разделенных на категории. Самые популярные из них: вычисления, сети, хранилища, базы данных, веб-службы, интернет вещей (IoT) и искусственный интеллект.

• **Управление:** Для связи всех компонентов используется платформа **Azure Resource Manager (ARM)**, которая позволяет группировать ресурсы и управлять ими как единым целым.

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

Вычислительные службы (Compute): Это основа Azure, позволяющая размещать код и приложения.

• **Виртуальные машины:** Создание серверов на Windows или Linux за считанные минуты.

• **Масштабируемые наборы:** Автоматическое увеличение количества серверов при росте нагрузки.

• **Azure Kubernetes Service (AKS):** Управление кластерами контейнеров.

• **Функции Azure:** Бессерверные вычисления, где вы платите только за время работы вашего кода.

Службы базы данных (Databases):

• **Azure SQL Database:** Полностью управляемая классическая реляционная база данных.

• **Azure Cosmos DB:** Глобально распределенная база данных для работы с огромными нагрузками и неструктурированными данными (NoSQL).

• **Базы для MySQL/PostgreSQL:** Готовые облачные версии популярных баз с открытым кодом.

• **Azure Synapse Analytics:** Хранилище для анализа данных объемом в несколько петабайт.

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

Если приложение работает на одной виртуальной машине, оно может «упасть» при любом сбое оборудования. Для защиты от этого в Azure есть несколько инструментов:

1. **Группа (набор) доступности (Availability Set):** Логическое объединение машин, которое гарантирует, что они будут физически стоять в разных стойках дата-центра.
2. **Домены отказа (Fault Domains):** Это группа серверов с общим питанием и сетевым коммутатором (одна стойка). Azure автоматически распределяет ваши машины по разным доменам отказа, чтобы поломка одной стойки не выключила всё приложение.
3. **Домены обновления (Update Domains):** Группы серверов для планового обслуживания. Когда Microsoft обновляет систему, она делает это по очереди в каждом домене, чтобы часть ваших серверов всегда оставалась включенной.
4. **Зона доступности (Availability Zone):** Это целые отдельные здания дата-центров внутри одного региона. У каждой зоны свое питание и охлаждение; если в одном здании случится пожар, машины в другом здании продолжат работать.

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

Azure HDInsight — это облачная служба, которая предоставляет управляемые кластеры **Apache Hadoop** для обработки колоссальных объемов данных.

- **Возможности:** Она позволяет выполнять запросы к структурированным и неструктурированным данным объемом в несколько петабайт. Система поддерживает популярные технологии, такие как **Spark**, **Hive** и **HBase**.
- **Интеграция с Интернетом вещей (IoT):** HDInsight часто используют для обработки потоковых данных, которые приходят в реальном времени с миллионов «умных» устройств (датчиков, счетчиков, автомобилей). Данные сначала собираются в «Центре Интернета вещей» (IoT Hub), а затем HDInsight анализирует их, чтобы бизнес мог мгновенно реагировать на изменения.

96. Система виртуализации с открытым кодом OpenNebula.

OpenNebula — это open-source платформа для управления виртуализированной инфраструктурой data-центров. Позволяет строить **private, public, hybrid и edge clouds** (IaaS). Основной фокус на KVM, но поддерживает VMware, LXC/LXD, Firecracker microVM.

Актуальная версия на январь 2026: **7.0.2** (вышла в декабре 2025), с улучшениями стабильности, бэкапами (Veeam, NetApp) и поддержкой новых ОС (RHEL 10, Debian 13).

Ключевые компоненты и возможности:

- Оркестрация хранения, сети, виртуализации и мониторинга.
- Поддержка гипервизоров: KVM (основной), VMware vCenter, LXC, Firecracker.
- Multi-tenancy, автоматическая эластичность, self-service портал.
- Hybrid cloud: интеграция с public провайдерами (AWS, Azure и др.).
- Kubernetes для контейнеров, поддержка GPU (NVIDIA vGPU для AI).
- Интерфейсы: EC2, OCCI, Sunstone (web-GUI).
- High availability, federation (много сайтов), edge computing.

Плюсы:

- Простота и гибкость: легче в установке и управлении, чем OpenStack.
- Нет vendor lock-in, полностью открытый код (Apache 2.0).
- Низкие затраты, подходит для enterprise (есть Enterprise Edition с поддержкой).
- Хорошая масштабируемость для private/hybrid облаков.
- Альтернатива VMware: миграция, похожие фичи (DRS-подобный scheduling, HA).

Минусы:

- Меньше фич в advanced networking/orchestration по сравнению с OpenStack или CloudStack.
- Community Edition без регулярных maintenance-релизов (только critical fixes).
- Для крупных сложных сред может потребоваться доп.настройка.

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

** DevOps** — это культура разработки ПО, которая объединяет процессы создания программ (Dev) и их последующей эксплуатации (Ops).

Причины появления:

• **Конфликт приоритетов:** Исторически разработчики стремятся как можно быстрее вносить изменения и добавлять функции, в то время как системные администраторы (Ops) ценят стабильность системы и боятся частых обновлений, которые могут «уронить» сервер.

• **«Гражданская война» отделов:** Разные команды имеют разные компетенции и используют разные инструменты, что порождает трения и замедляет работу.

• **Проблема сред:** Программа часто идеально работает у программиста («на моей машине всё ок»), но ломается при запуске на реальном сервере из-за различий в настройках и версиях ПО.

Цели и задачи:

• **Цели:** сократить время выхода продукта на рынок, снизить частоту отказов при выпуске новых версий и быстрее исправлять ошибки.

• **Задачи:** автоматизировать разработку и тестирование, согласовать действия всех команд и внедрить управление инфраструктурой как кодом.

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

Инфраструктура как код (IaC): Это подход, при котором настройки серверов, сетей и баз данных записываются в виде текстовых файлов на формальных языках (YAML, JSON, Ruby, Python и др.).

• Этот код выполняется компьютерами автоматически, что гарантирует **согласованность** настроек во всех средах (разработки, тестирования и эксплуатации).

• IaC является фундаментом облачных вычислений.

Изоляция вариантов развертывания: Чтобы избежать конфликтов, когда на одном сервере нужно запустить разные версии одной программы или когда настройки сервера не подходят приложению, применяют изоляцию.

• **Виртуальные машины (ВМ):** Позволяют быстро развернуть нужную среду и легко откатиться к сохраненному образу при сбое.

• **Контейнеры:** Более «легкий» способ, который запускается как обычный процесс в ОС, потребляет меньше ресурсов и позволяет достичь высокой плотности размещения приложений.

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

CI/CD — это сердце методологии DevOps.

• **CI (Непрерывная интеграция):** Разработчики вносят изменения в код несколько раз в день, после чего система автоматически собирает приложение и проверяет его тестами. Это позволяет находить ошибки на самых ранних этапах.

• **CD (Непрерывное развертывание):** После успешных тестов ПО автоматически передается в эксплуатацию. Это позволяет выпускать новые функции для пользователей очень часто, иногда по несколько раз в день.

Инструменты и конвейер (Pipeline): Для автоматизации этого процесса создается **конвейер DevOps** — последовательность автоматических шагов от написания кода до его мониторинга.

• **Популярные инструменты:** GitHub (хранение кода), Jenkins (сервер автоматизации), Docker/Kubernetes (контейнеры), Prometheus (мониторинг).

100. Реализация DevOps в AWS.

Amazon предлагает семейство сервисов для построения DevOps-конвейера:

• **AWS CodeCommit:** Облачный сервис для хранения исходного кода (на базе Git).

• **AWS CodeBuild:** Сервис, который берет код, компилирует его и запускает тесты.

• **AWS CodeDeploy:** Автоматически разворачивает готовое приложение на виртуальных машинах или локальных серверах.

• **AWS CodePipeline:** «Дирижер», который связывает все вышеперечисленные сервисы в единую цепочку.

• **IaC в AWS:** Реализуется через сервисы **CloudFormation** (сложные проекты), **OpsWorks** (на базе Chef) и **Elastic Beanstalk** (быстрое развертывание стандартных сайтов).

101. Реализация DevOPs в Azure.

Azure предлагает готовый набор инструментов **Azure DevOps** для полного цикла DevOps.

Основные компоненты:

- **Azure Boards** — планирование и трекинг задач: канбан-доски, бэклог, спринты, отчёты.
- **Azure Repos** — хранилище кода на Git (приватные репозитории, pull requests, code review).
- **Azure Pipelines** — CI/CD для любых языков и платформ. YAML-пайплайны, поддержка Azure, AWS, GCP, on-prem. Можно использовать свои или управляемые агенты Microsoft.
- **Azure Artifacts** — хранилище пакетов (NuGet, npm, Maven и др.), удобно для зависимостей в пайплайнах.
- **Azure Test Plans** — ручное и автоматизированное тестирование, трекинг результатов.

Плюсы:

- Всё в одном месте, глубоко интегрировано с другими сервисами Azure (App Service, AKS и т.д.).
- Хорошо работает с GitHub (можно комбинировать репозитории и Actions).
- Поддержка Jenkins и других внешних инструментов.

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

Альянс облачной безопасности (CSA) выделяет **12 основных угроз**, среди которых: утечка данных, взлом интерфейсов (API), кража учетных записей, DDoS-атаки и вредоносная деятельность инсайдеров (сотрудников).

Угрозы по отношению к виртуальным машинам (ВМ):

• **Побег из ВМ (VM Escape):** Самая опасная атака, когда злоумышленник из «гостевой» операционной системы прорывается к гипервизору и получает контроль над всем физическим сервером и другими ВМ на нём.

• **Угон гипервизора (Hyperjacking):** Установка скрытого вируса-руткита еще до загрузки гипервизора, что дает хакеру полный контроль над машиной.

• **Кража ВМ:** Злоумышленник может просто скопировать файл образа ВМ со всеми данными клиента, если управление файлами настроено небезопасно.

• **Мгновенные дыры (Instant-on Gaps):** Виртуальные машины часто включаются и выключаются; если ВМ долго была выключена, она может пропустить важные обновления безопасности и стать уязвимой сразу после включения.

• **Атаки между ВМ:** Если одна машина на сервере заражена, она может пытаться перехватить трафик соседей или атаковать их.