1. **Предпосылки появления облачных вычислений**

**Облачные вычисления** представляют собой динамически масштабируемый способ доступа к внешним вычислительным ресурсам в виде сервиса, предоставляемого посредством Интернета, при этом пользователю не требуется никаких особых знаний об инфраструктуре "облака" или навыков управления этой "облачной" технологией​.

Предпосылки:

**Рост:**

* производительности процессоров (CPU, GPU);
* Пропускной способности компьютерных сетей;
* Ёмкости оперативной и дисковой памяти.

**Виртуализация**

•VmWare, Microsoft, Citrix

**Интернет-сервисы**

•Amazon, SalesForce, Google

**Инструменты управления вычислительными ресурсами**

* Управление мощностями (перераспределение и выделение
* ресурсов
* **Сервисный подход к управлению ИТ**
* •ITIL, SLA
* **Сервисный подход к созданию ИС**
* •SOA, web services
* **Технологическое развитие**
* •Каналы связи
* •Интеллектуальные: сервера, сетевые устройства, устройства хранения данных

# **2. Обзор моделей (парадигм) распределенных вычислений.**

* Software as a Service (SaaS) - программное обеспечение как услуга.
* Platform as a Service (PaaS) - платформа как услуга.
* Infrastructure as a Service (IaaS) - инфраструктура как услуга.

|  |  |  |
| --- | --- | --- |
| **Модель** | **Основная деятельность (активности)** | **Примеры потребителей** |
| SaaS | Использует приложения/сервисы для автоматизации бизнес-процессов | Бизнес-пользователи, администраторы приложений |
| PaaS | Разрабатывает, тестирует, развертывает и управляет приложениями, развернутыми в облачном окружении | Разработчики приложений, тестировщики, администраторы |
| IaaS | Создает/устанавливает, управляет и мониторит сервисы для управления ИТ-инфраструктурой | Системные разработчики, администраторы, ИТ-менеджеры |

1. **Особенности облачных вычислений.**

¨Центральную роль на рынке облачных услуг играют поставщики и потребители этих услуг.

¨ИТ-услуги используются в качестве товара на открытом рынке без каких-либо технологических и юридических барьеров.

¨Наиболее распространенными причинами для внедрения облачных вычислений являются:

* Потребность в предоставляемых по требованию услуг хранения данных, вычислительной мощности, виртуальных инфраструктур для развертывания своих ИС и т.п.
* Возможность доступа к документам через Интернет, а также их обработка с использованием сложных приложений
* Оплата только за реально использованные ресурсы облака.
* Облачные услуги особенно привлекательны для разработчиков ПО, у которых нет инфраструктуры или которые не могут позволить себе дальнейшее расширение своей ИТ-инфраструктуры.
* Облачная инфраструктура надежна и доступна в любое время. Вычислительные службы облака являются высоконадежными, масштабируемыми и доступными не зависимо от места нахождения пользователя.
* Данные в облаке доступны отовсюду и представляются в стандартных форматах.
* Безопасность и надежность облачных вычислений будут продолжать развиваться, за счет расширения спектра используемых методов обеспечения безопасности.

1. **Основные характеристики (признаки) облачных вычислений**

Облачные вычисления – это модель предоставления повсеместного, удобного  и доступного по запросу доступа к разделяемому пулу конфигурируемых компьютерных ресурсов (виртуальных машин, систем хранения данных, приложений и сервисов), которые  могут быть быстро подготовлены и предоставлены  пользователю с минимальными усилиями управления  со стороны поставщика облачных услуг или взаимодействия с ним.​

* **Самообслуживание по требованию.**​

Потребитель по мере необходимости автоматически, без взаимодействия с поставщиком услуг, может самостоятельно определять и изменять вычислительные мощности, такие как серверное время, объем хранилища данных;​

* **Широкий (универсальный) сетевой доступ.**​

Вычислительные возможности доступны по сети через стандартные механизмы, что способствует широкому использованию разнородных (тонких или толстых) платформ клиента (терминальных устройств);​

* **Объединение ресурсов в пулы**. Конфигурируемые вычислительные и иные ресурсы поставщика объединены в единый пул для совместного использования распределенных ресурсов большим количеством потребителей;​
* **Мгновенная эластичность ресурсов**(мгновенная масштабируемость). Облачные услуги могут быстро предоставляться, расширяться, сжиматься и освобождаться исходя из потребностей потребителя; ​
* **Измеряемый сервис**(учет потребляемого сервиса и возможность оплаты реально использованных услуг. ​

**5. Модели развертывания облаков: частное облако, публичное облако, гибридное облако, общественное облако.**

Private cloud (частное облако) - инфраструктура, предназначенная для использования облачных вычислений в масштабе одной организации;

Community cloud (облако сообщества) - облачная инфраструктура, которая предназначена для исключительного использования облачных вычислений определенным сообществом потребителей от организаций, которые решают общие проблемы;

* Public cloud (публичное облако) - инфраструктура, предназначенная для свободного использования облачных вычислений широкой публикой;
* Hybrid cloud (гибридное облако) - это комбинация различных облачных инфраструктур (частных, публичных или сообществ), остающихся уникальными объектами, но связанных между собой стандартизованными или частными технологиями, которые обеспечивают возможность обмена данными и приложениями.

|  |  |  |
| --- | --- | --- |
| **Private**​ | **Частное**​ | 1. Реализация модели облачных вычислений на ресурсах, имеющихся в распоряжении одной компании, для обслуживания внутренних потребителей ​  2. Облачная инфраструктура функционирует целиком в целях обслуживания одной организации. Инфраструктура может управляться самой организацией или третьей стороной и может существовать как на стороне потребителя так и у внешнего провайдера. ​ |
| **Community**​  ​ | **Комму-нальное**​ | Облачная инфраструктура используется совместно несколькими организациями и поддерживает ограниченное сообщество, разделяющими общие принципы (например, миссию, требования к безопасности, политики, требования к соответствию <регламентам и руководящим документам>). Такая облачная инфраструктура может управляться самими организациями или третьей стороной и может существовать как на стороне потребителя так и у внешнего провайдера.​ |
| **Hybrid**​ | **Гибридное**​ | Облачная инфраструктура является композицией (сочетанием) двух и более облаков (частных, общих или публичных), остающихся уникальными сущностями, но объединенных вместе стандартизированными или частными (проприетарными) технологиями, обеспечивающими портируемость данных и приложений между такими облаками (например, такими технологиями, как пакетная передача данных для баланса загрузки между облаками). ​ |
| **Public**​ | **Публич-ное**​ | Облачная инфраструктура создана в качестве общедоступной или доступной для большой группы потребителей не связанной общими интересами, но, например, принадлежащими к одной области деятельности\*. Такая инфраструктура находится во владении организации, продающей соответствующие облачные услуги/ предоставляющей облачные сервисы. ​ \*) принадлежность к одной области деятельности/ индустрии может предполагать специфичные для этой индустрии приложения, потребность в которых испытывают организации, ведущие аналогичную деятельность или работающие на одном рынке. ​ |

**6. Модель предоставления услуг облачных вычислений: Software as a Service (SaaS), достоинства и недостатки.**

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

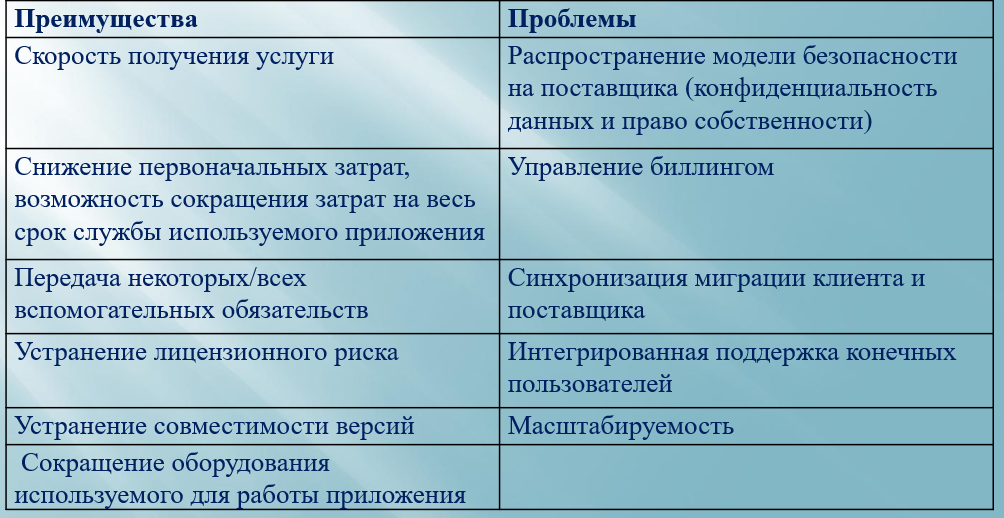
В модели SaaS:

* приложение приспособлено для удаленного использования;
* одним приложением могут пользоваться несколько клиентов;
* оплата за услугу взимается либо как ежемесячная абонентская плата, либо на основе суммарного объема транзакций;
* поддержка приложения входит уже в состав оплаты;
* модернизация приложения может производиться обслуживающим персоналом плавно и прозрачно для клиентов.

В облаке запускается один экземпляр службы, и несколько конечных пользователей могут получить доступ к службам одновременно.

Заказчикам не нужно вкладывать средства в серверы или лицензии на программное обеспечение, в то время как для провайдера затраты снижаются, поскольку необходимо размещать и поддерживать только одно приложение.

В настоящее время SaaS предлагают такие компании, как Google, Salesforce и Microsoft.



# **7.** **Модель предоставления услуг облачных вычислений: Platform as a Service (PaaS), достоинства и недостатки.**

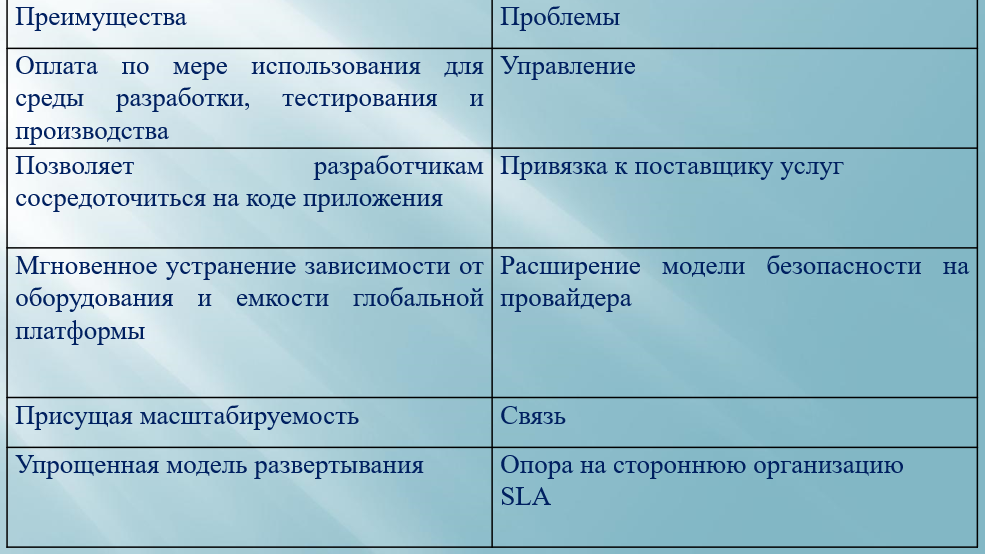
*PaaS - это предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги.*

В PaaS уровень программного обеспечения или среды разработки инкапсулируется и предлагается как услуга, на основе которой могут быть построены другие услуги более высокого уровня.

Заказчик может создавать свои собственные приложения в инфраструктуре поставщика. PaaS предоставляет вычислительную платформу и / или стек решений как услугу, часто использующую облачную инфраструктуру и поддерживающие облачные приложения.

Разработчики менее ограничены такими ресурсами, как память и вычислительная мощность. Они могут использовать существующие навыки работы с Microsoft Visual Studio и Microsoft. NET для создания привлекательных приложений и сервисов, размещаемых в облаке.

Созданы настраиваемые приложения и инструменты, которые повышают продуктивность разработчиков от имени всей инженерной организации. Google App Engine, Force.com и т. Д. - некоторые из популярных примеров PaaS



# **8. Модель предоставления услуг облачных вычислений: Infrastructure as a Service (IaaS), достоинства и недостатки.**

IaaS - это предоставление компьютерной инфраструктуры как услуги на основе концепции облачных вычислений.

IaaS состоит из трех основных компонентов:

* Аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование)
* Операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами)
* Связующее ПО (например, для управления системами)

IaaS предоставляет базовые вычислительные возможности и хранилище в качестве стандартизованных услуг по сети.

Серверы, системы хранения, пространство центра обработки данных, сетевое оборудование и т. д. объединенные в пул и доступны пользователям.

IaaS предоставляет инфраструктуру в составе: - сеть, хранилище данных , виртуальные машины и другие фундаментальные ресурсы, на которых потребитель может развертывать и запускать программное обеспечение.

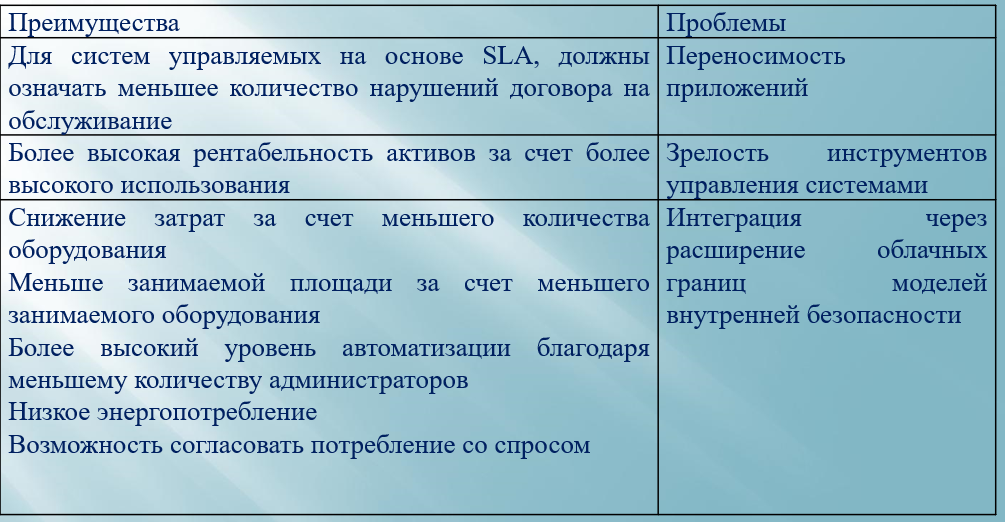
Это могут быть как приложения, так и операционные системы. Компании могут использовать IaaS  для оптимизации своей инфраструктуры.

Заказчик обычно развертывает собственное программное обеспечение в инфраструктуре, предоставляемой поставщиком IaaS. Типичными примерами являются Amazon, GoGrid, 3 Тера и т. д.

IaaS основана на технологии виртуализации, позволяющей пользователю оборудования делить его на части, которые соответствуют текущим потребностям бизнеса, тем самым увеличивая эффективность использования имеющихся вычислительных машин.

IaaS избавляет предприятия от необходимости поддержки сложных инфраструктур центров обработки данных, клиентских и сетевых инфраструктур, а также позволяет уменьшить связанные с этим капитальные затраты и текущие расходы.

Примеры. Первопроходцами в IaaS считается компания Amazon, которые на сегодняшний день предлагают два основных IaaS-продукта: EC2 ( Elastic Compute Cloud ) и S3 ( Simple Storage Service ).



**9. Проблемы облачных вычислений. Этические вопросы применения облачных систем**

Облачные вычисления можно рассматривать как новое явление, которое должно произвести революцию в способах использования Интернета, нужно быть очень осторожным в понимании рисков и проблем безопасности, возникающих при использовании облачных вычислений. К таким проблемам относятся:

* Реализация соглашения об уровне обслуживания (SLA)
* Управление данными в облаке
* Безопасность
* Совместимость (интероперабельность) облачных сервисов
* Управление энергоресурсами
* Мультиарендность ресурсов
* Консолидация серверов
* Надежность и доступность услуг
* Наличие общих облачных стандартов

***Этика:***

Облачные вычисления основаны на смене парадигмы с глубокими последствиями для компьютерной этики. Основными элементами этого перехода являются:

1. Контроль  за работой всей или части системы передается сторонним службам.

2. Данные хранятся на нескольких сайтах, управляемых несколькими организациями.

3. Несколько сервисов разных поставщиков взаимодействуют в сети.

# **10. Роль стандартизации API облачных сервисов. Организации, занимающиеся стандартизацией облачных сервисов. Стандарты облачных вычислений.**

К организациям, фокусирующимся на открытых стандартах облачных сервисов, относятся :

* OpenStack Foundation,
* Open Grid Forum и
* Open Group;

к отраслевым организациям стандартизации, которые организовали рабочие группы по облачным вычислениям, относятся:

* Distributed Management Task Force (DMTF)
* Storage Network Institute Association (SNIA).
* Рабочая группа по управлению облаком в DMTF
* Техническая группа по облачному хранению данных в SNIA;
* DMTF и Техническая группа по облачному хранению данных в SNIA определили стандартные интерфейсы для облачных вычислений.

к организациям стандартизации информационных технологий, которые предлагают одобренные или работающие стандарты облачных вычислений, относятся:

* National Institute of Standards and Technology (NIST)
* OASIS (Organization for the Advancementof Structured Information Standards).
* NIST опубликовал фактически признанное определение облачных вычислений;
* OASIS продвигает черновые варианты стандарта облачных вычислений;

к организациям, поддерживающим пользователей и предлагающим практические методы управления SLA, относятся:

* TM Forum
* Cloud Service Customer Council.

**Открытый формат виртуализации (OVF)**: устанавливает транспортный механизм для перемещения виртуальных машин с одной размещенной платформы на еще один.

**P2301**: это руководство по профилям облачной переносимости и взаимодействия (CPIP).

**P2302**: это стандарт межоблачного взаимодействия и федерации (SIIF).

**Открытый интерфейс облачных вычислений (OCCI)**: его цель - разработать API для задач управления облаком. API обеспечивает взаимодействие между облачной реализацией IaaS.

# **11. Препятствия для разработки облачных приложений и сервисов.**

Перечислим несколько наиболее очевидных препятствий:

 Доступность сервиса; что происходит, когда поставщик услуг не может выполнить поставку? Может ли такая крупная компания, как например General Motors, перенести свою ИТ-деятельность в облако и получить гарантии того, что ее деятельность не пострадает от перегрузки облака?

Частичным решением является заключение соглашения об уровне обслуживания (SLA).

Временное решение, но с отрицательными экономическими последствиями, - это избыточное выделение средств, то есть наличие достаточных ресурсов для удовлетворения наибольшего прогнозируемого спроса.

Привязка к поставщику; Как только клиент привязан к одному поставщику облачных услуг, становится трудно перейти к другому. Усилия по стандартизации в NIST пытаются решить эту проблему.

Конфиденциальность данных и возможность аудита; это действительно серьезная проблема.

Узкие места в передаче данных критичны для приложений с интенсивным использованием данных. Передача 1 ТБ данных в сети 1 Мбит / с занимает 8 000 000 секунд или около 10 дней; быстрее и дешевле использовать курьерскую службу и отправлять данные, перекодированные на некоторых носителях, чем отправлять их по сети. Высокоскоростные сети решат эту проблему в будущем, например, сеть 1 Гбит / с сократит это время до 8 000 секунд, или чуть более 2 часов.

Непредсказуемость производительности; это одно из последствий совместного использования ресурсов.

Эластичность, возможность быстрого увеличения и уменьшения. Необходимы новые алгоритмы управления распределением ресурсов и размещением рабочих нагрузок. Автономные вычисления, основанные на самоорганизации и самоуправлении, кажутся многообещающим направлением.

# **12. Понятие и принципы виртуализации. Область эффективной виртуализации Понятие виртуальной машины.**

Для описания работы вычислительной системы необходимы **три класса** фундаментальных абстракций:

интерпретаторы, память и каналы связи.

Им соответствуют следующие  физические реализации: процессоры для преобразования информации; первичная и вторичная память для хранения информации; и системы связи, позволяющие различным системам взаимодействовать друг с другом.

Программные системы управляют физическими ресурсами и преобразуют  физические реализации трех абстракций

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

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

Преимущества виртуализации:

1.Эффективное использование вычислительных ресурсов

2.Сокращение расходов на инфраструктуру

3.Снижение затрат на программное обеспечение.

4.Повышение гибкости и скорости реагирования системы.

5.Несовместимые приложения могут работать на одном компьютере

6.Повышение доступности приложений и обеспечение непрерывности работы предприятия

7.Возможности легкой архивации

8.Повышение управляемости инфраструктуры

Принципы виртуализации

В компьютерной системе имеется ряд интерфейсов, начиная с базового набора команд ЦПУ и кончая коллекцией АРI поставляемых с различным ПО современных систем  с промежуточным слоем.

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

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

**Виртуальная машина** *— это полностью изолированный программный контейнер, который работает с собственной ОС и приложениями, подобно физическому компьютеру. Виртуальная машина действует так же, как физический компьютер, и содержит собственные виртуальные (т.е. программные) ОЗУ, жесткий диск и сетевой адаптер*.

* 1. **История развития технологий виртуализации.**

Виртуализация серверов — размещение нескольких логических серверов в рамках одного физического.

Цели виртуализации:

* 1. Предоставить каждому пользователю изолированную среду исполнения приложений.
  2. Повысить гибкость использования ресурсов ЭВМ исполняемыми на ней приложениями.
  3. Повысить защищенность приложений друг от друга исполняемых на одной и той же ЭВМ.
  4. Повысить эффективность использования аппаратных средств ЭВМ.

Компания IBM была первой.

В 60-х годах прошлого века виртуализация представляла чисто научный интерес и была оригинальным решением для изоляции компьютерных систем в рамках одного физического компьютера.

* Примерно 1966. В кембриджском научном центре разработан эмулятор CP-40 для S/360-40, явившийся первой попыткой реализации полной программной виртуализации физической ЭВМ.
* 1967. На основе этой разработки была создана CP[-67]/CMS – virtual machine/virtual memory time-sharing operating system для S/360-67
* В начале 70-х гипервизор СР-67, был переработан в виде OS VM/370 для нового семейства машин System/370, выпущенного на рынок в 1972 г.
* Линейка машин System/370 в 1990-х годах была заменена компанией IBM линейкой System/390. Виртуализация в ОС MVS 390 была сохранена.
* В 2000 году IBM выпустила машины z-серии, поддерживающие 64-разрядное виртуальное адресное пространство при сохранении обратной совместимости с System/360 и поддержке виртуализации.
* Все эти системы фирмы IBM поддерживали виртуализацию на десятилетия раньше того момента, когда она приобрела популярность на машинах семейства x86.
* 1980-90 г.г. В это время основные работы в области виртуализации велись в направлении адаптации этой технологии для персональных ЭВМ и прежде всего для архитектуры Intel x86.
* В 1999 г. компания VMware представила технологию виртуализации систем на базе x86 получившую название VMware Virtual Platform. Первым продуктом реализующим новую технологию было ПО VMware WorkStation (гирервизор на основе хозяйской ЭВМ).
* 2005 г. VMware выпустила первое бесплатное ПО виртуализации десктопов - VMware Player.
* 2006 г. Выпустила ПО VMware ESX Server – первый гипервизор полной виртуализации серверов для архитектуры х86 (native hypervisor – “родной” гипервизор х86)
* Позднее в "битву“ включились такие компании как Parallels (ранее SWsoft), Oracle (Sun Microsystems), Citrix Systems (XenSourse). Корпорация Microsoft вышла на рынок средств виртуализации в 2003 г. с приобретением компании Connectiх, выпустив свой первый продукт Virtual PC для настольных ПК.
* 2011г. Сформулированы основные модели развертывания и признаки облачных вычислений. В основе облачных вычислений лежат технологии виртуализации.
  1. **Требования к архитектуре ЭВМ для поддержки виртуализации. Проблемы виртуализации платформы Intel x86**

В1974 году двое ученых Геральд  Попек  и Роберт  Голдберг  дали точный перечень тех условий,  которым  должна отвечать компьютерная архитектура, чтобы иметь возможность эффективно поддерживать виртуализацию.​

Такими требованиями являются:​

* 1.  **Безопасность**— у гипервизора должно  быть полное  управление виртуализированными ресурсами.​
* 2.  **Эквивалентность**— поведение программы на виртуальной машине  должно  быть идентичным поведению этой же программы, запущенной на реальном оборудовании. ​
* 3.  **Эффективность**— основная часть кода в виртуальной машине должна выполняться без вмешательства гипервизора.

**Проблемы виртуализации платформы Intel x86.**

Во-первых виртуализации мешают имеющиеся в x86 «иерархически защищенные домены» (hierarchical protection domain), которые еще называют «защитными кольцами» (Ring 0, Ring 1, Ring 2, Ring 3), обеспечивающие безопасность данных и функционирования в случае сбоев и ошибочного поведения. Эти кольца образуют иерархию, от наиболее привилегированного Ring 0, имеющего непосредственный доступ в память и ко всем ресурсам процессора, до наименее привилегированного Ring 3, на котором работают приложения. Наличие специальных шлюзов между кольцами ограничивает доступ с внешнего уровня на внутренний уровень. С точки зрения обеспечения безопасности эта архитектурная особенность полезна, но она создает барьер на пути к виртуализации.

Вторая сложность — несоответствие архитектуры x86 критерию достаточности виртуализации Попека-Гольдберга Еще в 70-е годы Попек и Гольдберг показали, что монитор виртуальных машин может быть создан для любого компьютера, если его команды, изменяющие состояние компьютера, входят в число привилегированных. Если условие удовлетворяется, то тогда команда, выполненная гостевой операционной системой, может быть перехвачена и перенаправлена в VMM. Но в системе команд IA-32 насчитывается 17 проблемных команд, которые подобным образом перехватить нельзя. При выполнении на разных уровнях такие команды имеют разную семантику.

Имеются команды способные изменить состояние процессора исполняемые в пользовательском режиме.

Для решения проблемы виртуализации архитектуры Intel используется дополнительное ПО, получившее название **монитор виртуальных машин**.

**15. Требования к гипервизору стратегия и основные подходы к обработке невиртуализируемых инструкций процессора.**

Паравиртуализация была разработана как альтернатива использованию двоичной трансляции при обработке невиртуализируемых инструкций процессора х86. При этом подходе требуется модификация гостевых операционных систем (для того, чтобы сделать возможными "гипервызовы" от виртуальной машины к гипервизору). Вместо выполнения (для обеспечения сохранности системного состояния) гипервизором (или монитором VMM) трансляции потенциально небезопасной инструкции гостевой операционной системы, для управления изменением состояния системы делается структурированный гипервызов от гостя к гипервизору.

Строгая реализация паравиртуализации дает повышение производительности на стандартном оборудовании х86 (за счет ликвидации дорогих операций при полной виртуализации и двоичной трансляции). Однако это делается за счет отсутствия поддержки немодифицированных гостевых операционных систем и миграции виртуальных машин обратно на физический сервер. Учитывая эти ограничения, программные продукты на основе паравиртуальных реализаций используют также и аппаратную виртуализацию (для работы немодифицированных операционных систем). Такой подход позволяет предоставить более широкую поддержку, охватывающую и устаревшие операционные системы, а также позволяет обновлять более новые операционные системы (используя те улучшения и повышение производительности, которые можно получить от паравиртуализации).

XenSource (недавно купленная компанией Citrix) была одной из компаний по реализации паравиртуализации, которая создала решение виртуализации с открытым кодом под названием Xen. Первоначальные версии Xen поддерживало только несколько модифицированных операционных систем. С выпуском версии Xen 3.0 (которая использует аппаратную поддержку виртуализации архитектур AMD-V и Intel VT) в гостевой виртуальной машине появилась возможность выполнения немодифицированной операционной системы Windows XP.

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

Центральный процессор является ядром:

* Реальной машины;
* Виртуальной машины.

*Возможности виртуализации процессора:*

* Позволяет «гостевой ОС» полагать, что она имеет эксклюзивный контроль над ЦПУ;
* В действительности ЦПУ разделяется между несколькими ОС.

*Для виртуализации памяти* ВМ необходимо обеспечить несколько уровней трансляции адресов:

* Витруальной памяти ВМ в физическую память ВМ;
* Физической памяти ВМ в физическую память реальной машины

Физические устройства внешней памяти:

* Жесткие диски;
* Трвердотельные диски;
* Подключаемые разделы внешних систем хранения.

Виртуальные диски:

* Файлы образов ВМ и файлы данных ВМ, располагаемые на физических устройствах внешней памяти;
* В случае «падения» ПО виртуализации, все виртуальные устройства внешней памяти теряются.

Могут быть виртуализированы следующие устройства:

* Сетевые карты;
* Порты COM, LPT, USB
* CD-ROM/DVD

*Может быть виртуализирован ввод/вывод на устройствах:*

* Видеоадаптер;
* Мышь и клавиатура.

***Виртуализация сети*** – это полное воспроизведение физической сети программным методом. Виртуальные сети аналогичны физическим сетям с точки зрения надежности и возможностей. Однако они обладают множеством дополнительных эксплуатационных преимуществ, таких как независимость от оборудования, быстрая инициализация, возможность развертывания без прерывания работы систем, автоматизированное обслуживание и поддержка как современных, так и устаревших приложений..

* + 1. **Два подхода к виртуализации процессора. Полная виртуализауция и паравиртуализация.**

***Полная виртуализация***. При использовании этого метода монитором VMM (для абстрагирования виртуальной машины от реального оборудования) создается и поддерживается полная виртуальная система. Этот подход позволяет выполнять в виртуальной машине операционную систему без всяких ее модификаций.

Преимуществом полной виртуализации и подхода с полным развязыванием физического оборудования и виртуальной машины является способность легко переносить виртуальные машины между серверами с различными физическими конфигурациями.

Такая гибкость достигается ценой потери производительности из-за накладных расходов на обслуживание состояний виртуальных машин и задержек при двоичной трансляции.

***Паравиртуализация***. Паравиртуализация была разработана как альтернатива использованию двоичной трансляции при обработке невиртуализируемых инструкций процессора х86. При этом подходе требуется модификация гостевых операционных систем (для того, чтобы сделать возможными "гипервызовы" от виртуальной машины к гипервизору). Вместо выполнения (для обеспечения сохранности системного состояния) гипервизором (или монитором VMM) трансляции потенциально небезопасной инструкции гостевой операционной системы, для управления изменением состояния системы делается структурированный гипервызов от гостя к гипервизору.

Строгая реализация паравиртуализации дает повышение производительности на стандартном оборудовании х86 (за счет ликвидации дорогих операций при полной виртуализации и двоичной трансляции). Однако это делается за счет отсутствия поддержки немодифицированных гостевых операционных систем и миграции виртуальных машин обратно на физический сервер. Учитывая эти ограничения, программные продукты на основе паравиртуальных реализаций используют также и аппаратную виртуализацию (для работы немодифицированных операционных систем). Такой подход позволяет предоставить более широкую поддержку, охватывающую и устаревшие операционные системы, а также позволяет обновлять более новые операционные системы (используя те улучшения и повышение производительности, которые можно получить от паравиртуализации).

**18. Возможные варианты реализации гипервизора (монитора виртуальных машин). Два вида реализации виртуальных машин**

Монитор виртуальных машин получил название Гипервизор. Различают: гипервизоры I и II типов.

**Гипервизор первого типа:** Используются не модифицированные экземпляры гостевых операционных систем,а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС,в роли которой выступает обычная операционная система.

Такая технология применяется, в частности, в VMware Workstation, VMware Server (бывший GSX Server), Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server (Hyper-V), Virtual Iron.​

Достоинства: ​

- не требуется модификации гостевой ОС ;​

- «прозрачность» виртуализации.​

Недостатки :​

— высокие дополнительные накладные расходы на используемые аппаратные ресурсы, ​

* отсутствие учета особенностей гостевых ОС, ​
* - меньшая, чем нужно, гибкость в использовании аппаратных средств​

**Гипервизор второго типа:**

Монитор виртуальных машин работает в рамках хозяйской ОС.

В процессе своей работы он модифицирует код гостевых ОС с целью обеспечения их виртуализации на уровне набора команд.

При этом код содержащий “чувствительные” команды переписывается на лету (Vmware WorkStation).

Накладные расходы составляют 10-15%

# **19. Два вида виртуализация средствами операционных систем.**

Имеется два вида виртуализации ОС:

Виртуализация с помощью монитора виртуальных машин (VMM – Virtual Machine Monitor) или **гипервизора**. **Гипервизор первого типа:** Используются не модифицированные экземпляры гостевых операционных систем,а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС,в роли которой выступает обычная операционная система.  **Гипервизор второго типа:** Монитор виртуальных машин работает в рамках хозяйской ОС. В процессе своей работы он модифицирует код гостевых ОС с целью обеспечения их виртуализации на уровне набора команд.При этом код содержащий “чувствительные” команды переписывается на лету (Vmware WorkStation). Накладные расходы составляют 10-15%

Виртуализация  средствами  базовой ОС или **контейнеризация**. Еще ее называют разделением на разделы (partition) или patitioning. Виртуализирует физический сервер на уровне ОС, позволяя запускать изолированные виртуальные серверы называемые Виртуальные Частные Серверы (Virtual Pivate Servers, VPS) или Контейнеры (Container, CT). Виртуализация на уровне операционной системы имеет минимальные накладные расходы и обеспечивает самую высокую степень консолидации, однако эта технология не позволяет запускать ОС с ядрами, отличными от ядра базовой ОС. **Накладные потери производительности контейнеров составляют 3%**

**20. Модель защиты памяти архитектуры Intel\_x86. Проблема одновременной работы гипервизора и гостевой ОС в режиме ядра. Использование кольца1 защиты памяти. Режимы защиты памяти при полной и паравиртуализации.**

**Модель памяти для платформ x86 —** способ указать предположения, которые должен сделать компилятор при генерации кода для платформ с сегментной адресацией памяти или со страничной памятью. Чаще всего термин употребляется при работе с различными устаревшими режимами платформы x86.

16-битная x86-архитектура, благодаря наличию четырёх сегментных регистров, позволяет одновременно иметь доступ к четырём сегментам памяти. Назначение сегментных регистров:

DS (data segment) — сегмент данных;

CS (code segment) — сегмент кода;

SS (stack segment) — сегмент стека;

ES (extra segment) — дополнительный сегмент.

Логический адрес на такой платформе принято записывать в виде сегмент:смещение, где сегмент и смещение задаются в шестнадцатеричной системе счисления.

В реальном режиме для вычисления физического адреса байта памяти происходит сдвиг влево на 4 разряда значения соответствующего сегментного регистра, а затем добавляется смещение.

**Проблема одновременной работы гипервизора и гостевой ОС в режиме ядра:**

Модификация ядра гостевой ОС выполняется таким образом, что в нее включается новый набор API, через который она может напрямую работать с аппаратурой, не конфликтуя с другими виртуальными машинами. При этом нет необходимости задействовать полноценную ОС в качестве хостового ПО, функции которого в данном случае исполняет специальная система, получившая название гипервизора (hypervisor).

**Кольца защиты** — архитектура информационной безопасности и функциональной отказоустойчивости, реализующая аппаратное разделение системного и пользовательского уровней привилегий. Структуру привилегий можно изобразить в виде нескольких концентрических кругов. В этом случае системный режим (режим супервизора или нулевое кольцо, так называемое «кольцо 0»), обеспечивающий максимальный доступ к ресурсам, является внутренним кругом, тогда как режим пользователя с ограниченным доступом — внешним.

Существуют четыре уровня привилегий, они пронумерованы от 0 (наиболее привилегированный уровень), до 3 (наименее привилегированный уровень), и три типа ресурсов, в отношении которых действуют механизмы защиты процессора: память, порты ввода / вывода и возможность выполнения некоторых инструкций. В любой момент, x86 процессор работает на определенном уровне привилегий, и от этого зависит, что может и чего не может сделать код. Уровни привилегий также часто называют кольцами защиты, которые изображаются в виде вложенных окружностей.

**Уровни привилегий программ**

**Кольцо 0** — ядро операционной системы, системные драйверы  
**Кольцо 1** — программы обслуживания аппаратуры, драйверы, программы, работающие с портами ввода/вывода компьютера  
**Кольцо 2** — системы управления базами данных, расширения операционной системы  
**Кольцо 3** — прикладные программы, запускаемые пользователем

Эффективное использование архитектуры колец защиты требует тесного взаимодействия между аппаратными средствами и ОС. ОС, разрабатывающиеся так, чтобы они работали на большом количестве платформ, могут иметь различную реализацию механизма колец на каждой платформе. Часто модель безопасности упрощают до двух уровней доступа: уровня «ядра» и уровня «пользователя», даже если аппаратные средства обеспечивают большую степень детализации уровней выполнения.

**21. Гипервизор первого типа достоинства и недостатки.**

На аппаратные средства устанавливается специализированная ОС (гипервизор) предназначенная для виртуализации на уровне набора команд аппаратной платформы (Vmware ESX).​ Такой монитор получил название “родного” (naitive) монитора виртуальных машин.​

**Достоинства данной технологии заключаются в:​**

* отсутствии потребности в хостовой ОС – ВМ, устанавливаются фактически на "голое железо", а аппаратные ресурсы используются более эффективно. ​

**Недостатки:​**

* необходимость поддержки в гипервизоре собственных драйверов внешних устройств, из-за чего возникают высокие дополнительные накладные расходы на используемые аппаратные ресурсы, ​
* отсутствие учета особенностей гостевых ОС, меньшая, чем нужно, гибкость в использовании аппаратных средств​

Используются не модифицированные экземпляры гостевых операционных систем,а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС,в роли которой выступает обычная операционная система.

Такая технология применяется, в частности, в VMware Workstation, VMware Server (бывший GSX Server), Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server (Hyper-V), Virtual Iron.​

Достоинства: ​

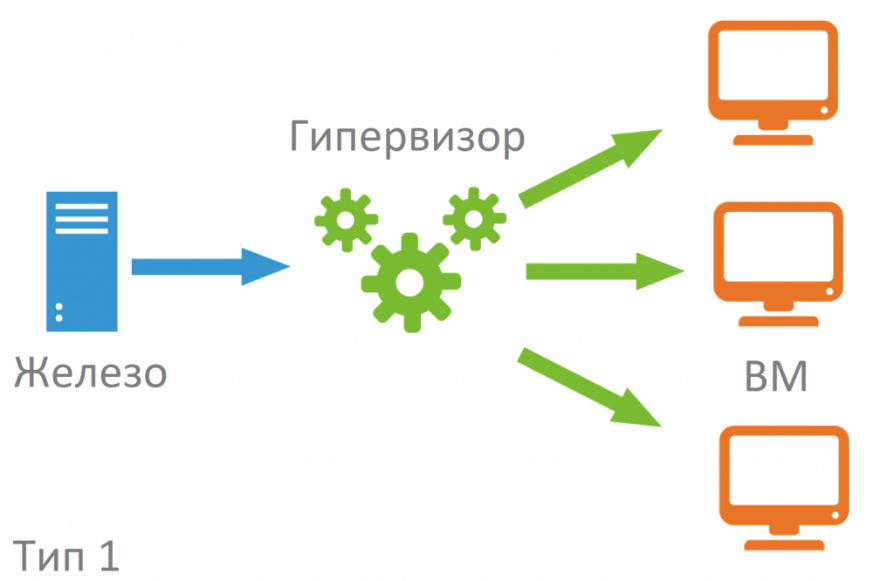
-  не требуется модификации гостевой ОС ;​

- «прозрачность» виртуализации.​

Недостатки :​

— высокие дополнительные накладные расходы на используемые аппаратные ресурсы, ​

* отсутствие учета особенностей гостевых ОС, ​
* - меньшая, чем нужно, гибкость в использовании аппаратных средств​



**22. Гипервизор 2-го типа достоинства и недостатки.**

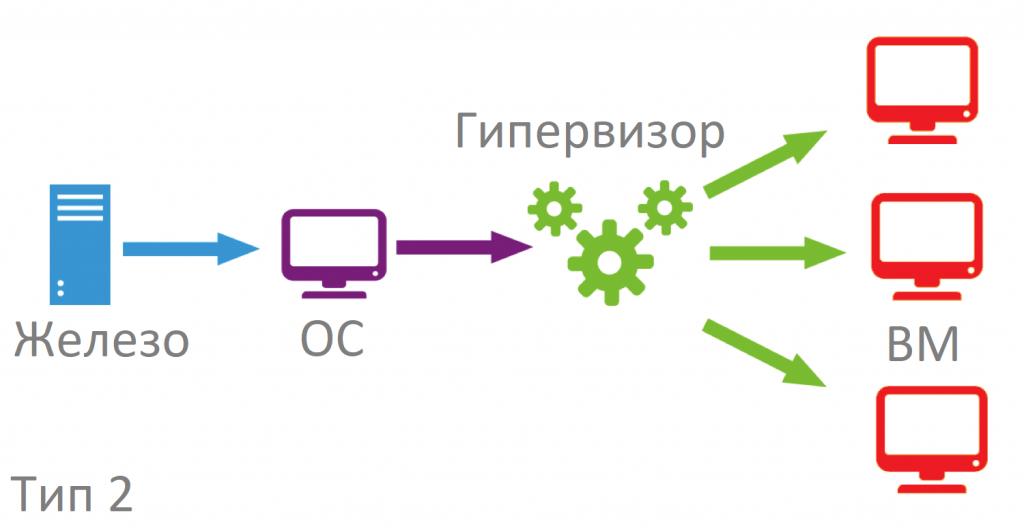
Монитор виртуальных машин работает в рамках хозяйской ОС.​

В процессе своей работы он модифицирует код гостевых ОС с целью обеспечения их виртуализации на уровне набора команд.​

При этом код содержащий “чувствительные” команды переписывается на лету (Vmware WorkStation).​

**Накладные расходы составляют 10-15%**​

Больше подходят для индивидуального использования, чем в качестве решений уровня предприятия.



* 1. **Паравиртуализация. Достоинства и недостатки.**

Используются не модифицированные экземпляры гостевых операционных систем, а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС, в роли которой выступает обычная операционная система.

* Модификация ядра гостевой ОС выполняется таким образом, что в нее включается новый набор API, через который она может напрямую работать с аппаратурой, не конфликтуя с другими виртуальными машинами. При этом нет необходимости задействовать полноценную ОС в качестве хостового ПО, функции которого в данном случае исполняет специальная система, получившая название гипервизора (hypervisor).

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

Недостатки — в сложности реализации подхода и необходимости создания специализированной ОС-гипервизора.

* 1. **Система виртуализации Xen.**

Эта технология позволяет гипервизору в хостовой системе управлять гостевой ОС посредством гипервызовов VMI (Virtual Machine Interface), что требует модификации ядра гостевой системы. На данный момент бесплатная версия Xen включена в дистрибутивы нескольких ОС, таких как Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. В середине августа 2007 года компания XenSource была поглощена компанией Citrix Systems.

Изначально Xen представлял собой самую развитую платформу,поддерживающую технологию паравиртуализации.

Open Source версия платформы Xen применяется в основном в образовательных и исследовательских целях.

Версии Xen включаются в дистрибутивы многих Linux-систем, что позволяет их пользователям применять виртуальные машины для изоляции программного обеспечения в гостевых ОС с целью его тестирования и изучения проблем безопасности, без необходимости установки платформы виртуализации​

Xen идеально подходит для поддержки старого программного обеспечения в виртуальной машине. Для более же серьезных целей в производственной среде предприятия необходимо использовать коммерческие платформы компании Citrix.

Особенности XEN

XEN имеет две наиболее значимые особенности: паравиртуализация и минимальность кода самого гипервизора.

Основной принцип паравиртуализации заключается в подготовке гостевых операционных систем путем незначительной модификации их ядра перед исполнением в виртуализированной среде. Это накладывает определенные ограничения на выбор используемых систем, так как от операционной системы требуется предоставление возможности внесения изменений в ее исходный код. Это касается как хост-систем, так и гостевых систем.

Вторым фактором, который относится к особенностям XEN, является незначительный объем кода самого гипервизора. Это достигается за счет выноса большего количества функций управления за пределы самого гипервизора.

**25 Система виртуализации Hyper-V.**

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

**Hyper-V** поддерживает разграничение согласно понятию *раздел*. **Раздел** — логическая единица разграничения, поддерживаемая гипервизором, в котором работают ОС. Каждый экземпляр гипервизора должен иметь один *родительский раздел*, с запущенной Windows Server 2008. Стек виртуализации запускается на родительском разделе и обладает прямым доступом к аппаратным устройствам. Затем родительский раздел порождает *дочерние разделы*, на которых и располагаются гостевые ОС. Дочерний раздел также может породить собственные дочерние разделы. Родительский раздел создает дочерние при помощи API-*гипервизора*, представленного в Hyper-V.

Hyper-V обеспечивает высокую производительность, практически равную производительности одной операционной системы, работающей на выделенном сервере.

Преимущества:

Низкие затраты на оборудование и программное обеспечение. Благодаря технологии виртуализации серверов Microsoft Hyper-V, на каждом физическом сервере может быть запущено несколько виртуальных серверов, используя одну лицензию на Windows Server Enterprise или Datacenter.

Высокая доступность и отказоустойчивость (новый уровень SLA). Кластер Hyper-V автоматически восстанавливает свою работоспособность менее чем за 5 минут даже в случае полного отказа одного из серверов.

Масштабируемость (готовность к росту бизнеса). Если возникает задача развёртывания нового сервиса или приложения, вы просто за считанные минуты создаёте новую виртуальную машину и разворачиваете необходимые приложения.

Снижение затрат на обслуживание

1. **Система виртуализации VMware.**

В 1998 году VMware запатентовала свои программные техники виртуализации и с тех пор выпустила немало эффективных и профессиональных продуктов для виртуализации различного уровня: от VMware Workstation, предназначенного для настольных ПК, до VMware ESX Server, позволяющего консолидировать физические серверы предприятия в виртуальной инфраструктуре.

В процессорах на базе x86 содержатся 17 особых инструкций, создающих проблемы при виртуализации, из-за которых операционная система отображает предупреждающее сообщение, прерывает работу приложения или просто выдает общий сбой.

Для преодоления этого препятствия компания VMware разработала адаптивную технологию виртуализации, которая "перехватывает" данные инструкции на этапе создания и преобразует их в безопасные инструкции, пригодные для виртуализации

* **VMware Workstation** – платформа, ориентированная на desktop-пользователей и предназначенная для использования разработчиками ПО, а также проффесинальными пользователями.
* **VMware Player** – бесплатный "проигрыватель" виртуальных машин на основе виртуальной машины.
* **VMware ESX Server** – это гипервизор, разбивающий физические серверы на множество виртуальных машин. VMware ESX является основой пакета VMware vSphere и входит во все выпуски VMware vSphere.
* **VMware Fusion** – настольный продукт для виртуализации на платформе Mac от компании Apple.
* **VMware vSphere** – комплекс продуктов, представляющий надежную платформу для виртуализации ЦОД.
* VMware vSphere включает ряд компонентов, преобразующих стандартное оборудование в общую устойчивую среду,:
* Службы инфраструктуры — это компоненты, обеспечивающие всестороннюю виртуализацию ресурсов серверов, хранилищ и сетей, их объединение и точное выделение приложениям.
* Службы приложений — это компоненты, предоставляющие встроенные элементы управления независимо от их типа или ОС.
* VMware vCenter Server предоставляет центральную консоль , которая поддерживает масштабируемость для управления крупными средами ЦОД.

1. **Виртуализация на уровне ОС достоинства и недостатки.**

 Виртуализирует физический сервер на уровне ОС, позволяя запускать изолированные виртуальные серверы называемые Виртуальные Частные Серверы (Virtual Pivate Servers, VPS) или Контейнеры (Container, CT).

Виртуализация на уровне операционной системы имеет минимальные накладные расходы и обеспечивает самую высокую степень консолидации, однако эта технология не позволяет запускать ОС с ядрами, отличными от ядра базовой ОС.

Накладные потери производительности контейнеров составляют 3%

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

Недостатки — реализация только однородных вычислительных сред.

**28 Виртуализация хранилищ данных. Классификация виртуальных хранилищ. Достоинства и недостатки виртуальных хранилищ данных.**

Цель ***виртуализации хранилищ*** как одного из компонентов программного хранилища – повышение производительности без приобретения дополнительного оборудования для хранения данных.

Классификация виртуализации хранилищ описывает пять различных типов виртуализации хранилища: блочная, дисковая, лента (как носитель, накопитель и библиотека), файловая система, виртуализация файлов

**К преимуществам облачных хранилищ можно отнести:**

Обеспечивает очень быстрое и надежное хранилище для вычислений и обработки данных

Предоставляет администратору системы хранения большую гибкость в управлении хранилищем для конечных пользователей.

Предоставляет возможности для оптимизации использования хранилища и консолидации серверов, а также для выполнения миграции файлов без прерывания работы.

Позволяет каждому виртуальному серверу запускать свою собственную операционную систему, и каждый виртуальный сервер также может быть перезагружен независимо друг от друга.

Снижает стоимость, поскольку требуется меньше оборудования

Использует ресурсы для экономии эксплуатационных расходов

Уменьшает количество серверов.

Параллельно можно запускать больше приложений.

**Недостатки облачных хранилищ**

объем дискового пространства виртуального хранилища ограничен,

миграция данных занимает много времени.

**29 Блочные виртуальные хранилища данных. Достоинства и недостатки.**

В основе файловой системы лежит иерархическая структура: корневая запись, от которой отходят данные о файлах и их атрибутах. Все они, в свою очередь, организованы в удобную структуру каталогов – зная имя того или иного документа, доступ к нему можно получить, щелкнув мышью по его имени. С ними можно осуществлять любые операции – открывать, изменять, переименовывать, удалять, копировать, перемещать в другую папку.

В блочном хранилище структура размещения та же, но все попадающие туда файлы делятся системой на блоки, каждому из которых присваивается свой идентификатор. С его помощью система собирает файлы в случае надобности.

Преимущества:

Каждая пользовательская среда находится отдельно, за счет чего можно рассортировывать данные и обеспечить отдельный доступ к ним.

БХ обеспечивает повышенную производительность: благодаря хост-адаптеру шины, который разгружает процессор и освобождает его ресурсы для выполнения других задач.

Недостатки:

Оно дороже, и им трудно управлять, поскольку работа с блоками создает дополнительную нагрузку на базу данных.

Оно, как и файловое, ограничено в объеме.

Для чего подходит: для работы с корпоративными базами данных

**30 Файловые виртуальные хранилища данных. Достоинства и недостатки.**

В основе файловой системы лежит иерархическая структура: корневая запись, от которой отходят данные о файлах и их атрибутах. Все они, в свою очередь, организованы в удобную структуру каталогов – зная имя того или иного документа, доступ к нему можно получить, щелкнув мышью по его имени. С ними можно осуществлять любые операции – открывать, изменять, переименовывать, удалять, копировать, перемещать в другую папку.

Файловое хранилище может быть двух видов: физическим и виртуальным. В первом случае данные сохраняются на жестком диске, во втором – на виртуальном. Последний имеет намного больший объем чем жесткий, а еще туда можно настроить удаленный доступ. В качестве примера можно привести Dropbox, «Облако Mail.Ru», «Google Диск», «Яндекс. Диск» и другие аналогичные им сервисы.

Преимущества:

Простая и понятная структура.

В таком хранилище легко ориентироваться, искать нужные документы.

Недостатки:

Ограниченность в объеме, по мере заполнения которого падает скорость доступа, а вместе с ней и производительность.

Для чего подходит: для работы с небольшими объемами разны данных.

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

Цель ***виртуализации хранилищ*** как одного из компонентов программного хранилища – повышение производительности без приобретения дополнительного оборудования для хранения данных.

**Сетевое хранилище (NAS)** — это выделенное файловое хранилище, которое позволяет нескольким пользователям и разнородным клиентским устройствам получать данные с централизованного дискового пространства. Пользователи в локальной сети (LAN) получают доступ к общему хранилищу через стандартное соединение Ethernet. Устройства NAS настраиваются и управляются с помощью служебной программы на основе браузера. Каждый NAS находится в локальной сети как независимый сетевой узел, определяемый своим собственным уникальным IP-адресом.

Что больше всего характеризует NAS, так это простота доступа, высокая емкость и довольно низкая стоимость. Устройства NAS предоставляют инфраструктуру для консолидации хранилища в одном месте и для поддержки таких задач, как архивирование и резервное копирование, а также задач облачного уровня.

**DAS (Direct Attached Storage** - **система хранения данных с прямым подключением**) –запоминающее устройство для хранения данных, которое подключено непосредственно к серверу, или к рабочей станции, как правило, через интерфейс по протоколу SAS без помощи сети хранения данных. DAS часто называют «островами информации».

Системы типа DAS состоят из накопителя (например, жёсткого диска), соединенного с компьютером адаптером контроллера шины. Между ними нет сетевого устройства (концентратора, коммутатора или маршрутизатора), и это основной признак DAS. При этом накопители могут быть как внутренними, так и внешними. Другие серверы могут получить доступ только через сервер владельца. Простейший случай DAS-системы — это один диск внутри сервера или ПК.

**Сеть хранения данных (англ. Storage Area Network, SAN)** — представляет собой архитектурное решение для подключения внешних устройств хранения данных, таких как дисковые массивы, ленточные библиотеки, оптические приводы к серверам таким образом, чтобы операционная система распознала подключённые ресурсы как локальные.

SAN характеризуются предоставлением так называемых сетевых блочных устройств (обычно посредством протоколов Fibre Channel, iSCSI или AoE), в то время как сетевые хранилища данных (англ. Network Attached Storage, NAS) нацелены на предоставление доступа к хранящимся на их файловой системе данным при помощи сетевой файловой системы (такой как NFS, SMB/CIFS, или Apple Filing Protocol). При этом категоричное разделение SAN и NAS является искусственным: с появлением iSCSI началось взаимное проникновение технологий с целью повышения гибкости и удобства их применения.

**32. Виртуализация сети. Методы виртуализации VLAN, VIP, VPN. Достоинства и недостатки виртуализации сетей.**

**Виртуализация сети** - это метод объединения доступных ресурсов в сети. Это возможно за счет разделения доступной полосы пропускания на каналы, каждый из которых независим от других, и каждый может быть назначен (или переназначен) определенному серверу или устройству в режиме реального времени. Виртуализация сети может объединять различные физические сети в единую логическую сеть (виртуализация внешней сети) или предоставлять сетевые функции для раздела ОС (виртуализация внутренней сети).

*Виртуальная локальная сеть (VLAN)* - это метод создания независимых сетей поверх общей (физической) сети. VLAN контролирует взаимодействие между разными сетями в одной физической сети. VLAN - это общая черта всех современных коммутаторов Ethernet. Коммутатор Ethernet - это устройство, которое соединяет несколько сегментов сети и позволяет сетевым устройствам эффективно обмениваться данными. Коммутатор позволяет создавать несколько виртуальных сетей, а также изолирует каждый сегмент от других. VLAN - это самый безопасный метод создания независимых логических сетей в общей (физической) сети. Никакое взаимодействие с другими устройствами невозможно, даже если они подключены к одной и той же физической сети. Средствами VLAN можно в общей сетевой инфраструктуре облака создать несколько специализированных виртуальных сетей, например, сети ВМ, сеть СХД, сеть связи с Интернет, сеть управления узлами виртуализации.

*VIP* – это IP-адрес из частного адресного пространства, который назначается сетевому устройству и не связан с конкретным компьютером или сетевой картой (NIC). В основе механизма VIP лежит метод сетевой трансляции адресов (NAT – Network Address Translation). Входящие пакеты отправляются на VIP, но перенаправляются (с помощью механизма NAT) на фактический сетевой интерфейс принимающего хоста или хостов. VIP используется в технологиях виртуализации. Он поддерживает высокую доступность и балансировку нагрузки, когда у нескольких систем есть общее приложение, и они могут принимать трафик, перенаправленный сетевым устройством. Виртуальный IP-адрес устраняет зависимость хоста от отдельных сетевых интерфейсов, и поэтому даже в случае отказа компьютера или сетевого адаптера VIP-адрес может оставаться доступным, поскольку другой сетевой адаптер может отвечать на соединение.

*Назначение VPN* - гарантировать конфиденциальность на незащищенном сетевом канале. Обычно он используется как средство, позволяющее удаленному сотруднику подключиться к сети организации. Обычно это делается с помощью специального программного обеспечения (например, Cisco VPN Client). Программное обеспечение помогает в инициации соединения, и после того, как соединение установлено, все взаимодействие с ресурсами в сети происходит так, как если бы компьютер физически подключен к той же сети, но все это зависит от политик безопасности, применяемых в сети.

1. **Виртуализация десктопов. Преимущества и недостатки.**

Виртуализация отделяет ПО от оборудования (+). Такое разделение означает, что ПО помещается в отдельный контейнер, чтобы изолировать его от ОС.

Виртуализация рабочих столов

Виртуализация рабочего стола – отделение рабочего стола с его ОС, приложениями и пользовательскими данными от базовой вычислительной системы. Виртуализация рабочего стола предоставляет пользователям операционную среду, независимую от их локальных физических систем. Он состоит из серверов, ПО виртуализации на серверах и виртуального образа на рабочих столах. ЦОД и пользователь могут получить доступ к своей среде рабочего стола из любого рабочего места в сети (+).

Виртуализация представлений (рабочих мест)

Виртуализация представлений подразумевает эмуляцию интерфейса пользователя. Т.е. пользователь видит приложение и работает с ним на своём терминале (+), хотя на самом деле приложение выполняется на удалённом сервере, а пользователю передаётся лишь картинка удалённого приложения.

С ростом масштабов организаций, использование в ИТ-инфраструктуре пользовательских ПК вызывает ряд сложностей: (це все + выходит:)

большие операционные издержки на поддержку компьютерного парка;

сложность, связанная с управлением настольными ПК;

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

техническое сопровождение пользователей;

установка и обновление лицензий на ПО и техническое обслуживание;

резервное копирование и т.д.

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

Преимущества виртуализации десктопов

Пользователи могут получить доступ к своим настольным средам из любого места через LAN, WAN или широкополосный доступ. Обновления приложений можно делать регулярно. Обеспечивает повышенную безопасность благодаря централизованному администрированию. Это быстро и надежно; упрощает резервное копирование пользовательских данных.

К недостаткам виртуализации настольных компьютеров можно отнести:

Количество клиентских машин конечных пользователей, которые необходимы в сети, фиксировано. Тонкие клиенты иногда так же дороги, как и отдельные компьютеры. Пропускной способности должно быть достаточно, чтобы избежать перегрузки в локальной сети и обеспечить хорошую обработку и просмотр приложений с рабочего стола. Следует ограничивать количество ОС, которые могут поддерживать ПО виртуализации рабочих столов.

Достоинства: Эффективность использования ресурсов, Масштабируемость, Простые резервное копирование и миграция, Гибкость

Недостатки: Проблема в распределении ресурсов при высокой загрузке, Сложная настройка

**34.Виртуализация приложений. Достоинства и недостатки.**

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

Приложение исполняется без процедуры инсталляции в традиционном ее понимании и может запускаться прямо с внешних носителей (с флэш-карт, из сетевых папок). Данная технология позволяет использовать на одном компьютере, а точнее в одной и той же ОС несколько несовместимых между собой приложений одновременно.

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

Виртуализация приложений включает в себя выполнение следующих шагов:

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

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

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

Преимущества виртуализации приложений

Нет необходимости устанавливать приложение. Развертывание приложений происходит быстрее. Более простое и эффективное управление приложением. Использование виртуализации приложений приводит к снижению затрат. Повышенная безопасность

Единственный серьезный недостаток виртуализации приложений - постоянная поддержка полосы пропускания.

**35.Использование сервис-ориентированной архитектуры в облачных приложениях и службах.**

Применение СОА позволяет решить проблемы создания и функционирования РИС: Проблему изоляция бизнес-логики; Проблему обеспечения совместимости компонентов; Проблему избыточности.

SOA – логический способ организации программных систем для предоставления пользователям услуг через опубликованные и обнаруживаемые интерфейсы. SOA – это не продукт, а архитектурный стиль. С помощью SOA бизнес-логику можно разложить на четко определенную службу многократного использования, и ею может пользоваться каждый разработчик. SOA предоставляет бизнес-функции в виде сервисов, которые будут использоваться приложениями, чтобы разработчики могли сосредоточиться на основной бизнес-логике

Осн понятия СОА:

Сервис – повторно-исполняемая задача в рамках бизнес процесса.

Процесс (бизнес задача) – композиция, составленная из отдельных сервисов. Процесс определяет логику взаимодействия сервисов, независимо от их реализации.

SOA описывается как совокупность сервисов, реализуемых в виде компонентных объектов, систематизированных на основе обмена сообщениями (message-passing). Общим примером сообщений, которыми обмениваются сервисы является XML сообщение.

Для каждого сервиса принято определять: поставщика сервиса (компонент сервис), потребителя сервиса (компонент клиент); брокера - компонент, обеспечивающий взаимодействие поставщика и потребителя.

Организация работы приложения в сервис-ориентированной архитектуре

С многоуровневой точки зрения приложение можно описать трехуровневой архитектурой: Уровень интерфейса пользователя, Уровень бизнес логики, Уровень данных

Сначала создается новый промежуточный слой сервисов, (вводится соотв. API между уровнем пользовательского интерфейса и уровнем бизнес-логики). Затем реализуется разделение функций на уровне бизнес-логики. На следующем этапе преобразования основное внимание уделяется функциональным возможностям разделения доступа к данным.

Элементы сервисно-ориентированной архитектуры

Сервис в SOA имеет четыре свойства:

1. Он логически представляет собой бизнес-деятельность с заданным результатом.

2. Он самодостаточен.

3. Это черный ящик для потребителей.

4. Он может состоять из других базовых сервисов.

Преимущества сервис-ориентированной архитектуры: Возможность быстрее и проще создавать бизнес-приложения. Более легкое обслуживание и обновление. Гибкость и расширяемость. Снижение совокупной стоимости владения

SOA – является основой построения надежных распределенных систем, которые в качестве услуг предоставляют функциональные возможности, с дополнительным акцентом на слабые связи между взаимодействующими сервисами. Принцип слабой связи подразумевает, что взаимодействующие программные компоненты имеет минимальное знание друг о друге: они находят информацию, которая им нужна для взаимодействия непосредственно перед взаимодействием с помощью брокера.

Веб-сервис - это интерфейс, описывающий службу и загружаемый с удаленного провайдера сервиса или общего реестра сервисов. Веб-сервисы хорошо подходят в качестве строительных блоков SOA-среды.

1. **Микросервисная архитектура в облачных приложениях**

Микросервисы часто противопоставляются традиционной монолитной архитектуре. Монолит означает, что компоненты продукта взаимосвязаны и взаимозависимы. Если перестает работать один - все остальные тоже «отваливаются». Предшественником микросервисной архитектуры является SOA, которая также разделяет бизнес-логику на компоненты. Микросервисная архитектура - частный случай SOA c набором более строгих правил.

У микросервисов есть особые свойства, они же преимущества:

* Гетерогенность: возможность построить систему с помощью разных языков программирования и технологий;
* Децентрализованное управление данными: каждый микросервис содержит свой набор данных, доступный другим микросервисам только через соответствующий интерфейс;
* Независимость инфраструктуры: каждый микросервис - независимая единица, поэтому вносить изменения и разворачивать его можно независимо от других;
* Масштабируемость: чтобы увеличить производительность системы, нужно расширить только те сервисы, которые в этом нуждаются.

Микросервисная архитектура выигрывает у монолита:

Проще изменить один из микросервисов и сразу внедрить его, чем изменять весь монолит и перезапускать инфраструктуру целиком;

Новые разработчики легче включаются в работу - для этого им не нужно изучать систему целиком, можно работать только над своей частью;

Микросервисы не зависят от какой-либо платформы, поэтому внедрять новые технологии проще, чем в монолит.

Недостатки:

Сложность начальной разработки и создания инфраструктуры, т.к. нужно предусмотреть независимость одного микросервиса от сбоя в другом компоненте;

Для распределенной системы сложно поддерживать строгую согласованность: общие части системы нужно либо складывать в общую библиотеку, но тогда при изменении этой библиотеки нужно будет перезапускать и все зависимые микросервисы, либо хранить общий код в каждом из микросервисов, что сложнее поддерживать;

1. **Бессерверная технология реализации облачных приложений и служб.**

Бессерверная архитектура – это способ создания и запуска приложений и сервисов без необходимости управления инфраструктурой. Приложение по-прежнему работает на серверах, но управление этими серверами AWS полностью берет на себя. Это избавляет от необходимости заниматься выделением ресурсов, масштабированием и обслуживанием серверов для запуска приложений, баз данных и систем хранения данных.

Для реализации бессерверных архитектур можно использовать облачные сервисы, такие как: AWS Lambda (основан на микросервисах); Amazon API Gateway; Amazon DynamoDB (NoSQL-СУБД); реляционная СУБД Aurora.

Другие облака:

Microsoft Azure Functions

Google Functions

IBM OpenWhisk

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

Интернет облако представляется пользователю как массивный кластер серверов, размещенных в одном или нескольких ЦОДах. Эти сервера предоставляют ресурсы для выполнения веб-сервисов или распределенных приложений по запросам. Облачная платформа формируется динамически из серверов, программного обеспечения и ресурсов баз данных. Сервера в облаке могут быть как физическими, так и виртуальными машинами.

Интерфейс пользователя обеспечивает доступ к службам обработки запросов на оказание облачных услуг.

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

Цели создания платформ облачных вычислений – предоставление ресурсов по запросам пользователей обеспечивая: масштабируемость, виртуализацию ресурсов, эффективность использования ресурсов, надежность предоставления ресурсов, безопасность предоставления разделяемых ресурсов и разделяемый доступ к ЦОДам.

Включает следующие компоненты: ЦОДы – центры обработки данных; управление снабжением ресурсами облака; управление доступом к облаку; каталоги предоставляемых облаком сервисов; мониторинг производительности и безопасности облака.

Архитектура облака разработана как 3-х уровневая: Уровень инфраструктуры (IaaS); Уровень платформы (PaaS); Уровень приложений (SaaS).

**Проблемы** архитектурного дизайна облаков:

Проблема доступности и блокировки данных.

Конфиденциальность данных и их защита.

Непредсказуемая производительность

Распределенные хранилища данных и широко распространенные ошибки программного обеспечения.

Массштабируемость, Интероперабельность и стандартизация.

Лицензирование ПО и разделение репутации.

1. **Возможности облачных систем, используемые для создания собственных служб и приложений.**

Масштабируемость

Использование одноразовых ресурсов вместо фиксированных серверов

Автоматизация

Уменьшение связности между компонентами

Сервисы вместо серверов

Гибкие варианты хранения данных. Базы данных

Устранение единственных точек отказа

Оптимизация стоимости

Кеширование

Безопасность

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

Облачные вычисления основаны на смене парадигмы с глубокими последствиями для компьютерной этики. Основными элементами этого перехода являются:

1. Контроль за работой всей или части системы передается сторонним службам.

2. Данные хранятся на нескольких сайтах, управляемых несколькими организациями.

3. Несколько сервисов разных поставщиков взаимодействуют в сети.

Перечислим несколько наиболее очевидных препятствий:

Доступность сервиса; что происходит, когда поставщик услуг не может выполнить поставку? Может ли такая крупная компания, как например General Motors, перенести свою ИТ-деятельность в облако и получить гарантии того, что ее деятельность не пострадает от перегрузки облака?

Привязка к поставщику; Как только клиент привязан к одному поставщику облачных услуг, становится трудно перейти к другому. Конфиденциальность данных и возможность аудита; это действительно серьезная проблема.

Узкие места в передаче данных критичны для приложений с интенсивным использованием данных.

Непредсказуемость производительности; это одно из последствий совместного использования ресурсов.

Эластичность, возможность быстрого увеличения и уменьшения. Необходимы новые алгоритмы управления распределением ресурсов и размещением рабочих нагрузок. Автономные вычисления, основанные на самоорганизации и самоуправлении, кажутся многообещающим направлением.

1. **Архитектурные принципы и свойства веб сервисов** REST.

Системы, которые, как ожидается, будут расти со временем, должны быть построены на основе масштабируемой архитектуры. Такая архитектура может поддерживать рост пользователей, трафика или размера данных без снижения производительности. Архитектура должна обеспечивать линейный рост масштаба, когда добавление дополнительных ресурсов приводит, по крайней мере, к пропорциональному увеличению способности обслуживать дополнительную нагрузку. Рост объемов потребляемых ресурсов должен обеспечивать экономию на масштабе, а стоимость должна соответствовать степени увеличения произодительности системы и ценности ее для бизнеса.

Технология REST (Representational State Transfer - «передача состояния представления») практически вытеснила дизайн интерфейса, основанный на SOAP и WSDL, из-за более простого стиля проектирования.

Используется всеми публичными облачными системами.

Свойства архитектуры, которые зависят от ограничений, наложенных на REST-системы. Выполнение этих ограничительных требований обязательно для REST-систем. Накладываемые ограничения определяют работу сервера в том, как он может обрабатывать и отвечать на запросы клиентов. Действуя в рамках этих ограничений, система приобретает такие желательные свойства как производительность, масштабируемость, простота, способность к изменениям, переносимость, отслеживаемость и надёжность.

Существует шесть ограничений (5 обязательных и 1 опционное):

1. Модель клиент-сервер

2. Отсутствие состояния

3. Кэширование

4. Единообразие интерфейса

5. Слои

6. Код по требованию (необязательное ограничение)

Если сервис-приложение нарушает любое из этих ограничительных условий, данную систему нельзя считать REST-системой.

Архитектура REST основана на четырех принципах:

Идентификация ресурса посредством URI, который предоставляет глобальное адресное пространство для поиска ресурсов и сервисов.

Унифицированный интерфейс – GET извлекает текущее состояние ресурса в некотором представлении. POST устанавливает новое состояние ресурса.

Информативные сообщения – Ресурсы отделены от их представления таким образом, что их содержимое может быть доступно в различных форматах (например, HTML, XML, текст, RDF, JPEG).

Взаимодействие через гиперссылки – Для обмена существуют различные технологии (например, переименование URI, cookies и скрытые поля формы).

Состояние может быть помещено в ответное сообщение, чтобы указать допустимое будущее состояние взаимодействия

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

Масштабируемость – это способность повышать производительность системы наиболее рациональным способом. На практике это обычно достигается путем добавления вычислительных ресурсов, как аппаратных, так и / или программных. Однако, переписать существующий код в большинстве случаев является довольно радикальным решением, поэтому в 9 из 10 случаев компании останавливают увеличение количества серверов или расширение возможностей существующей серверной системы.

Вертикальное масштабирование происходит за счет увеличения показателей производительности, используемых ресурсов ВМ (экземпляров EC2): Увеличение размера жесткого диска; Увеличение производительности процессора; Увеличение размера ОП и д.р.

Этот способ масштабирования в конечном итоге имеет предел, и это не всегда рентабельный или высокодоступный подход. Однако его очень легко реализовать, и его может хватить для многих случаев использования, особенно в краткосрочной перспективе.

Горизонтальное масштабирование происходит за счет увеличения количества ресурсов, например, добавления дополнительных жестких дисков в массив хранения или добавления дополнительных серверов для поддержки приложения.

Это отличный способ создания интернет-приложений, использующих эластичность облачных вычислений.

Не все архитектуры способны масштабироваться за счет распределения своей рабочей нагрузки на несколько ресурсов, имеются следующие возможные сценарии масштабирования в зависимости от архитектуры облачных приложений:

приложения без сохранения состояния;

перераспределение нагрузки на несколько узлов;

компоненты без сохранения состояния;

компоненты с отслеживанием состояния;

с привязкой к сеансу;

использование распределенной обработки.

Рест

Отделение потребности интерфейса клиента от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, а упрощение серверной части улучшает масштабируемость.

Применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузки и распределённого кэширования.

Микросервисы

Масштабируемость: чтобы увеличить производительность системы, нужно расширить только те сервисы, которые в этом нуждаются.

Более масштабируема

СОА

Расширяемость возникает тогда, когда необходимо внедрить совершенно новую бизнес-услугу; все, что нужно сделать, это собрать соответствующие сервисы, которые уже существуют.

функциональные возможности сервиса могут быть расширены или сужены, поскольку при этом описание сервиса динамически меняется и, соответственно, изменяются и запросы.

Менее масштабируема

1. **Обеспечение обмена данными между компонентами облачных систем с помощью ESB (Enterprise Service Bus).**

* ESB – выполняет функции посредничества при:
  + Обмене сообщениями;
  + Регистрации;
  + Маршрутизации;
  + Ведении журналов;
  + Аудите;
  + Управлении транзакционной целостностью.
* Современные приложения редко работают изолированно; приложение не может сделать что-либо значимое без взаимодействия с другими приложениями.
* Сервис-ориентированная архитектура интегрирует приложения для совместной работы и ускоряет их работу, разбивая приложение на части, которые могут быть объединены друг с другом.
* Шина обеспечивает возможность подключения конечных точек друг к другу без прямого контакта друг с другом. Это упрощает связь для конечных точек, поскольку они должны соответствовать стандартным коммуникационным интерфейсам - шине. (Это с любой технической шиной, а не только с ESB).
* ESB может расширять существующую инфраструктуру обмена сообщениями, предоставляя инструментарий для создания и развертывания инфраструктурного уровня систем, построенных на основе обмена сообщениями.
* Примеры таких систем могут включать сервисы маршрутизации и преобразования, сервисы журналирования и т.д.
* Это может быть простым расширением нижележащей инфраструктуры, построенной на обмене сообщениями, либо мостом между разными продуктами и даже технологиями. Для доступа к приложениям или сервисам, не поддерживающим обмен сообщениями, могут использоваться адаптеры.

1. **Общая архитектура SOA облачного приложения но основе ESB.**



**Сервисная шина предприятия** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *enterprise service bus*, *ESB*) — [связующее программное обеспечение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D1%83%D1%8E%D1%89%D0%B5%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), обеспечивающее централизованный и унифицированный событийно-ориентированный обмен сообщениями между различными [информационными системами](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) на принципах [сервис-ориентированной архитектуры](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B8%D1%81-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0).

* ESB – выполняет функции посредничества при:
  + Обмене сообщениями;
  + Регистрации;
  + Маршрутизации;
  + Ведении журналов;
  + Аудите;
  + Управлении транзакционной целостностью.
* Современные приложения редко работают изолированно; приложение не может сделать что-либо значимое без взаимодействия с другими приложениями.
* Сервис-ориентированная архитектура интегрирует приложения для совместной работы и ускоряет их работу, разбивая приложение на части, которые могут быть объединены друг с другом.
* Модель SOA (потребители службы вызывают поставщиков службы) может показаться простой, но возникают две важные проблемы:
  + Как потребителю найти провайдера службы, которую он хочет вызвать?
  + Как потребитель может быстро и надежно вызвать службу в медленной и ненадежной сети?
* Оказывается, существует прямое решение обеих этих проблем – подход, называемый Enterprise Service Bus (ESB – сервисная шина предприятия).
* ESB упрощает вызов службы как для потребителя, так и для поставщика, управляя всеми сложными взаимодействиями между ними.
* ESB не только упрощает вызов службы приложениями (или их частями), но и помогает им передавать данные и распространять уведомления о событиях.

**Три ключевых преимущества:**

* Шина обеспечивает возможность подключения конечных точек друг к другу без прямого контакта друг с другом. Это упрощает связь для конечных точек, поскольку они должны соответствовать стандартным коммуникационным интерфейсам - шине. (Это с любой технической шиной, а не только с ESB).
* ESB предоставляет одно место для получения некоторых ключевых показателей конечной точки: частота, доступность и производительность.
* ESB имеет тенденцию предоставлять более одного интерфейса связи. Однако разработчику нужно только выбрать самый простой способ получить и получить данные с шины.

1. **Шаблоны взаимодействия облачных приложений с помощью службы ESB.**

Варианты реализации могут быть скомпонованы в шаблоны в зависимости от возможностей связующего ПО, поддерживающего эти архитектурные шаблоны

* Дизайн ESB реализует множество признанных шаблонов проектирования и спецификаций стандартов:
  + **Простой сервис-фасад**
  + **Сложный сервис-фасад**
  + **Клиентский доступ**

1. **Контейнеры. Возникновение контейнеров. Контейнерная виртуализация в Linux.**

Альтернативным подходом к изоляции приложений являются контейнеры. Лидером в продвижении контейнерных технологий является компания Parallels, представив продукт Virtuozzo.

Virtuozzo был создан изначально для Linux, в 2005 г. появилась версия для Windows , примерно тогда же компания Sun реализовала Solaris Containers для своей ОС (и появился термин «контейнеры»).

Название Virtuozzo порой бывает синонимом всего этого направления виртуализации, поскольку остальные средства намного уступают ему в популярности.

Экземпляры пространств пользователя (часто называемые контейнерами или зонами) с точки зрения пользователя полностью идентичны реальному серверу, но они в своей работе используют один экземпляр ядра операционной системы.

Для linux-систем, эта технология может рассматриваться как улучшенная реализация механизма chroot.

Ядро обеспечивает полную изолированность контейнеров, поэтому программы из раз­ных контейнеров не могут воздействовать друг на друга.

**Наиболее распространены сейчас OpenVZ, LXC, FreeBSD jail и Solaris Containers**

1. **Существующие варианты контейеризации.**
2. **OpenVZ** позволяет на одном физическом сервере запускать множество изолированных копий операционной системы, называемых «виртуальные частные серверы» (Virtual Private Servers, VPS) или «виртуальные среды» (Virtual Environments, VE).

Поскольку OpenVZ базируется на ядре Linux, в отличие от виртуальных машин (напр. VMware, Parallels Desktop) или паравиртуализационных технологий (напр. Xen), в роли «гостевых» систем могут выступать только дистрибутивы Linux.

OpenVZ является базовой платформой для Virtuozzo — проприетарного продукта Parallels

1. **LXC** (Linux Containers) — система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров операционной системы Linux на одном узле.

LXC основана на технологиях namespaces и cgroups, входящей в ядро Linux, начиная с версии 2.6.29.

1. **Механизмы изоляции ОС, используемые при контейнеризации.**

В 1979 году в UNIX был добавлен системный вызов chroot() — с целью обеспечить изоляцию и предоставить разработчикам отдельную от основной системы площадку для тестирования. chroot представляет собой сокращение от change root, что дословно переводится как «изменить корень». С  помощью системного вызова chroot() и соответствующей команды можно изменить корневой каталог

Попытки усовершенствовать механизм chroot и обеспечить более надёжную изоляцию предпринимались неоднократно: так, в частности, появились такие известные технологии, как FreeBSD Jail и Solaris Zones (про него не было в презентации).

Механизм виртуализации в системе FreeBSD, позволяющий создавать внутри одной операционной системы FreeBSD несколько независимо работающих FreeBSD на том же ядре операционной системы, но совершенно независимо настраиваемых с независимым набором установленных приложений.

В ядре Linux изоляция процессов была усовершенствована благодаря добавлению новых подсистем и новых системных вызовов:

* + Механизм пространств имен (на основе использования различных привилегий, назначаемых разным пользователям)
  + Механизм групп контроля (cgroups) (ограничивают количество ресурсов, которые разрешено использовать каждому отдельному контейнеру)

Принцип работы cgroups заключается в следующем:

определённые процессы помещаются в группу, которую затем «встраиваются» в подсистемы.

1. **Docker – как средство быстрого развертывания контейнеров с ПО.**

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений в контейнерах.

Docker:

* позволяет отделить приложение от его инфраструктуры.
* дает возможность запускать практически любое приложение, безопасно как изолированное в контейнере.

Docker дает возможность достаточно просто упаковать приложения вместе со всеми его зависимостями, а затем продолжить их разработку, тестирование и регулярное использование в совершенно другой среде.

Благодаря тому, что Docker позволяет упаковывать каждый компонент и его зависимости по-отдельности, решаются следующие задачи:

* + **Исчезают конфликтующие зависимости**: нужно запустить один веб-сайт на PHP 4.3 и еще один на PHP 5.5? Нет проблем в случае, если вы запускаете каждую версию PHP в отдельном контейнере Docker.
  + **Нет потерянных зависимостей**: установка приложения в новой среде осуществляется при помощи Docker и поэтому в контейнере вместе с приложением упакованы все зависимости.
  + **Нивелируется различие платформ**: переход от одного дистрибутива к другому уже не является проблемой. Если в обеих системах работает Docker, то один и тот же контейнер будет этих системах выполняться без всяких проблем.

1. **Архитектура Docker. Процесс создания образа. Запуск контейнера.**

**Базовая Архитектура Docker**

* Client Docker (CLI)
* Server Docker (Docker Engine);
* REST API

Сервер, по сути являющийся долго работающим приложением, называемым демоном

REST API, определяющее интерфейсы для взаимодействия, которые могут использовать другие программы.

Консольный (CLI) клиент использует Docker REST API для управления или взаимодействия с демоном Docker при помощи скриптов или непосредственных консольных команд.

Элементы архитектуры

* Docker Клиент (REST API)
* Docker Engine (демон)
* Docker File
* Образы (images)
* Реестры (registries)

Docker Hub

* Контейнеры

Образ контейнера – это «исходный» код на основе которого строятся контейнеры.

Образ является «шаблоном», а контейнер копия с этого шаблона.

**Как создать Docker образ?**

* запуск команды с указанием:
  + добавление файла или директории
  + создание переменной окружения
  + указания что запускать, когда запускается контейнер этого образа
  + инструкции хранятся в файле Dockerfile

Docker считывает этот Dockerfile, выполняет находящиеся там инструкции и возвращает конечный образ.

Важно понимать, что Docker ориентирован на контейнеризацию приложений, а не ОС.

* На основе образа диска с использованием в качестве описания:
  1. Имеющегося контейнера (интерактивное создание);
  2. Docker-файла (создание на основе docker-файла).

Запустить простой контейнер помощью команды: $ sudo docker run –i -t ubuntu /bin/bash

1. **Контейнеры Windows Server и Hyper-V.**

**Контейнеры Windows Server**

Обеспечивают изоляцию приложений благодаря изоляции процессов и пространств имен.

Контейнер Windows Server использует одно ядро ОС совместно с хостом, на котором он работает, и всеми остальными контейнерами на этом узле.

**Hyper-V**

Открывают более широкие возможности изоляции по сравнению с контейнерами Windows Server, так как каждый контейнер запускается в виртуальной машине со специальной легковесной версией ОС.

В этой конфигурации каждый контейнер использует свою копию ядра, изолированную от других контейнеров, но при этом он обладает характеристиками обычного контейнера (быстрое развёртывание, использование библиотеки шаблонов Docker, stateless, мощные возможности по управлению).

1. **Использование контейнеров в Microsoft Azure.**

Azure Container Service — облачный сервис контейнеризации на базе Docker Swarm ( кластеризация) и DC/OS (Apache Mesos). (DC / OS -это операционная система центра обработки данных, а также распределенная система. Операционная система основана на Apache Mesos distributed kernel.)

Этот сервис позволяет тысячам заказчиков по всему миру эффективно развёртывать масштабные решения с использованием контейнеров популярных форматов Docker и Mesos.

Windows Server для контейнеризации использует одну из самых популярных систем в индустрии — Docker на базе ОС Linux (это было в слайде про ажур ¯\\_(ツ)\_/).

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

Облачное хранилище данных (англ. cloud storage) — модель онлайн-хранилища, в котором данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиентам.

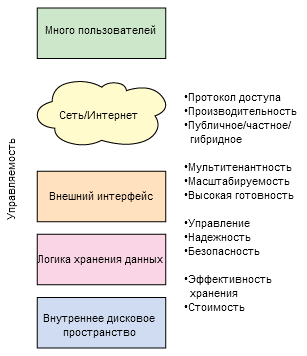
Плюсы:

1. Возможность бесплатно хранить определённый объём файлов, у каждого провайдера есть свои условия использования и ограниченное свободное место;
2. Оплата услуг идёт только за фактическое использование хранилища, а не за аренду всего сервера;
3. Уменьшение общих издержек на ИТ инфраструктуру организации, т.е. клиентам нет необходимости заниматься приобретением, поддержкой и обслуживанием собственной ресурсов по хранению данных;
4. Неограниченный доступ к данным отовсюду;
5. резервирование и все процедуры, связанные с сохранностью и целостностью данных, производятся самим провайдером облачного центра, клиент в этот процесс никак не вовлекается;
6. гибкость облачных хранилищ, сервис подходит практически, под все операционные системы, как персональные компьютеры, так и мобильные;
7. синхронизация файлов и доступ к облачным файлам, через операционную систему пользователя, зависит от самого клиента облака, поэтому нужно обращать внимание при выборе облака, на его готовность к работе с десктопным клиентом.

Минусы:

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

Обобщенно облачная архитектура хранения данных представляет собой внешний интерфейс, который предоставляет API для доступа к накопителям



|  |  |
| --- | --- |
| **Характеристика** | **Описание** |
| Управляемость | Способность управлять системой при наличии минимальных ресурсов |
| Метод доступа | Протокол, через который предоставляются услуги облачного хранения данных |
| Производительность | Измеряется пропускной способностью и временем задержки |
| Мультитенантность | Поддержка множества пользователей (арендаторов) |
| Масштабируемость | Возможность постепенного наращивания для удовлетворения новых требований или обработки повышенной нагрузки |
| Готовность данных | Измеряется временем безотказной работы системы |
| Управление | Возможность управлять системой — в частности, выбирая стоимость, производительность или другие характеристики |
| Эффективности хранения | Мера эффективности использования накопителей |
| Стоимость | Мера стоимости хранения данных (обычно в долларах за гигабайт) |

1. **Методы доступа к облачным хранилищам данных.**

Для доступа к ОХД используются следующие методы доступа:

1. API Web-сервисов в том числе RESTfull-API (Amazon S3), Windows Azure™ , Mezeo Cloud Storage Platform;

Использование API Web-сервиса, требует интеграции с приложенем, поэтому используются и общие методы доступа:

1. API На основе файлов:

NFS;

Common Internet File System(CIFS);

FTP;

1. API На основе блоков:

iSCSI

1. Другие API:

WebDAV (Web-based Distributed Authoring and Versioning)

1. **Облачные модели хранения данных.**
   1. Частное облако ЧД (IBM, Parascale и Cleversafe (которая предлагает программное обеспечение и/или оборудование для внутреннего облака));
   2. Публичное облако ХД (Amazon и Nirvanix);
   3. Гибридное облако ХД (Nirvanix, Egnyte).
2. **Типы облачных хранилищ**

Все типы облачных хранилищ делятся на:

1) блочные (диски);

2) файловые;

3) «базовые» (СУБД);

4) объектные.

|  |  |  |
| --- | --- | --- |
| **Тип облачного ресурса** | **Тип данных** | **Примечание** |
| Диск | Файлы, папки | Можно управлять файловой системой диска, разбивать его на логические диски и т. д.; доступное дисковое пространство ограничено размером предоставленного диска |
| Папка | Файлы, папки | Нельзя управлять файловой системой и логической структурой дискового пространства; размер хранилища ограничен тарифом, а не размером диска |
| Набор данных | Числа, строки и другие атомарные данные | Приложение должно «знать» логические связи между этими данными, чтобы их правильно использовать |
| Объект | Набор связанных данных | Как правило, этот набор соответствует чему-то из реального мира: пользователю, документу, организации, товару и т. д. |

Между собой они различаются степенью агрегации данных — если дисковые и файловые хранилища содержат в себе атомарные данные, то в СУБД и, тем более, в объектных хранилищах данные собраны в связанные структуры, которыми можно оперировать на более высоком, прикладном, уровне Невозможно заранее сказать, какой из названных типов хранилищ данных является наилучшим — его выбор зависит от характера решаемых прикладных задач и применяемого программного обеспечения.

1. **Обеспечение производительности облачных хранилищ данных. Протокол FASP.**

Главная задача облачной системы хранения данных ― это перемещение данных между пользователем и удаленным поставщиком облачных услуг.

Проблема кроется в том, что TCP управляет потоком данных на основе подтверждения приема пакетов из удаленного узла. Потеря или задержка пакетов приводит к применению мер по ограничению скоплений пакетов с дополнительным ограничением производительности во избежание глобальных сетевых проблем.

Amazon с помощью Aspera Software решила эту проблему, исключив TCP при передаче данных.

Был разработан новый протокол *Fast and Secure Protocol* (FASP™) на базе UDP. UDP позволяет управлять заторами протоколу прикладного уровня FASP/

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

1. **Методы обеспечения высокой готовности облачных хранилищ данных. Алгоритм рассредоточения данных IDA.**

Когда поставщик облачных услуг хранит данные пользователя, он должен иметь возможность возвратить эти данные пользователю по требованию.

Существуют интересные новые схемы обеспечения высокой готовности, такие как рассредоточение информации.

* Компания Cleversafe, использует алгоритм рассредоточения информации (Information Dispersal Algorithm - IDA) для повышения доступности данных перед лицом физических отказов и простоев сети.
* Он позволяет "нарезать" данные с помощью кодов Рида-Соломона для их восстановления в случае потери части данных.

IDA позволяет настраивать количество долей данных, так чтобы заданный объект данных можно было разрезать на **четыре** доли при одном допустимом сбое или **на 20 долей** при восьми допустимых сбоях.

IDA позволяет восстанавливать данные из подмножества исходных данных при некоторых накладных расходах на коды ошибок

1. **Apache Hadoop. Распределенная файловая система HDFS. Репликации HDFS. Технологии работы с большими данными. MapReduce.**

Является одной из основополагающих технологий «[больших данных](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5)».

Разработан на [Java](https://ru.wikipedia.org/wiki/Java) в рамках вычислительной парадигмы [MapReduce](https://ru.wikipedia.org/wiki/MapReduce), согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.

По состоянию на 2014 год, проект состоит из четырёх модулей —

1. Hadoop Common ([связующее программное обеспечение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D1%83%D1%8E%D1%89%D0%B5%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) — набор инфраструктурных программных библиотек и утилит, используемых для других модулей и родственных проектов),
2. HDFS ([распределённая файловая система](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)),
3. YARN(система для планирования заданий и управления кластером)
4. Hadoop MapReduce(платформа программирования и выполнения распределённых MapReduce-вычислений).

В HDFS вместо таблицы используется специальный сервер — сервер имён (NameNode), а данные разбросаны по серверам данных (DataNode).

данные разбиты на блоки (обычно по 64Мб или 128Мб)

HDFS имеет классическую Unix-овскую древовидную структуру директорий, пользователей с триплетом прав, и даже схожий набор консольных команд:

Выполняет 2 главные задачи: запись метаданных и хранение собственно данных.

**MapReduce** обеспечивает параллельные локальные вычисления на машинах кластера Hadoop с блоками данных, что обеспечивает быструю обработку больших объемов данных.

Принцип работы MapReduce состоит из двух шагов: map и reduce. Вот как происходит процесс:

**Map**: получение входных данных главным узлом (master node) -> разделение информации на части -> передача данных рабочим узлам (worker node).

**Reduce**: получение главным узлом ответа от рабочих узлов -> формирование результата.

1. **Apache Spark. Назначение, архитектура и возможности.**

[Apache Spark](http://spark.apache.org/) - альтернатива Hadoop. представляет собой движок для распределённой обработки данных.

Apache Spark обычно использует компоненты Hadoop, такие как HDFS и YARN для своей работы, при этом сам в последнее время стал популярнее, чем Hadoop

Архитектура Spark включает в себя 3 главных компонента:

* 1. хранилище данных;
  2. API;
  3. менеджер кластера.

В специфических задачах, построенных на многократных обращениях к одному набору данных, «чистый» Spark работает быстрее Hadoop до **30 раз**.

Например, это интерактивный дата-майнинг и итерационные алгоритмы, которые активно используются в системах машинного обучения.

Наиболее часто используется поверх Hadoop, заменяя модуль MapReduce и добавляя большее количество операций.

1. **Средства работы с BigData в Azure и AWS.**

Windows Azure позволяет быстро создавать Hadoop-кластеры на базе полнофункционального Apache Hadoop и затем:

с помощью Windows Azure PowerShell и интерфейса командной строки Windows Azure можно без труда интегрировать HDInsight в существующие рабочие процессы аналитики, чтобы получать действенные прогнозы от HDInsight путем интеллектуального анализа данных в Microsoft Excel.

Amazon EMR предоставляет управляемую инфраструктуру Hadoop, которая способна эффективно, быстро и экономично обрабатывать большие объемы данных на динамически масштабируемых инстансах Amazon EC2. В EMR можно также запускать другие известные распределенные инфраструктуры, включая [Apache Spark](https://aws.amazon.com/emr/details/spark/), [HBase](https://aws.amazon.com/emr/details/hbase/), [Presto](https://aws.amazon.com/emr/details/presto/) и [Flink](https://aws.amazon.com/blogs/big-data/use-apache-flink-on-amazon-emr/). С Amazon EMR можно использовать разные хранилища данных, в том числе Amazon S3, распределенную файловую систему Hadoop (HDFS) и Amazon DynamoDB.

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

**Google Drive**: Облачное хранилище делит пространство между Google Drive, Gmail и Google Photo.

В сервисе можно хранить не только документы, но и фотографии, музыку, видео и многие другие файлы – всего 30 типов. Всё очень удобно и привычно для пользователей Google сервисов.

Максимальный обьем файла 5 Тбайт.

Доступен в веб - браузерах, Windows, Mac OS, Android, iOS и др. 15 u, бесплатно

**OneDrive**: базирующийся на облачной организации интернет-сервис хранения файлов с функциями файлового обмена.

Интегрирован в Office 365 и в службу Windows Live OneDrive. 5 u, бесплатно

**DropBox**: 256 - битное шифрование AES и шифрование SSL; Лучшая в своем классе технология синхронизации; Интеграция с Microsoft Office 365; Неограниченное восстановление файлов и журнал версий; Ссылки доступа с паролем и сроком действия; Настраиваемые уровни доступа; Управление уровнями доступа и пр. 2u, бесплатно

Главным недостатком Dropbox можно считать подход к выбору папок для синхронизации. Фактически приложение следит за содержимым только одной папки Dropbox.

Dropbox не в полной мере заботится о конфиденциальности данных пользователя и даже напрямую участвует в глобальной системе слежке PRISM. Однако, не все согласны с этим.

**Mega**: шифрует весь контент прямо в браузере с помощью алгоритма AES; пользователи могут передавать друг другу файлы в зашифрованном виде, при этом все данные хранятся в «облаке»; ключи доступа к файлам не публикуются в открытом доступе, а распространяются по схеме Friend-to-Friend, между доверяющими друг другу пользователями. одним из самых выгодных облачных сервисов (50 гб в сумке у папы); кроме того, важное отличие Mega от других подобных сервисовэто то, что он защищает личные данные пользователя.

**Яндекс Диск** Позволяет пользователям хранить свои данные на серверах в облаке и передавать их другим пользователям в интернете. Работа построена на синхронизации данных между различными устройствами. В настоящее время регистрация пользователей доступна всем. Яндекс.Диск может выступать в качестве службы облачного сервиса, интегрируясь в офисный пакет Microsoft Office, а недавно появилась возможность автоматической загрузки фото и видеофайлов с цифровых камер и внешних носителей информации на Яндекс. Диск. 10 гб

1. Управление пользовательскими данными в хранилище «Dropbox» через REST API.

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

Инфраструктура GAE включает в себя компоненты:

* GFS – Google File System;
* MapReduce – модель и сервис распределенных вычислений
* BigTable - проприетарная высокопроизводительная база данных, построенная на основе Google File System (GFS);
* Chubby - Бесплатный онлайн словарь;
* Sheduler – планировщик на основе cron

Компоненты GAE обеспечивают следующую функциональность:

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

1. **Облачная платформа GAE - основные приложения, достоинства и недостатки.**

Gmail, Google Drive, Календарь, Документы, таблицы, презентации, Hangouts и тд

***Достоинства***

1. Для GAE быстро и легко разрабатывать
2. Не требует усилий по обслуживанию
3. Очень щедро выделены ежедневные бесплатные ресурсы
4. Вполне разумные цены при превышении квот
5. Лёгкое и быстрое выкладывание

***Недостатки***

1. Некоторые приложения невозможно реализовать из-за ограниченности функциональных возможностей
2. Контролируемое окружение не позволяет использовать некоторые стандартные библиотеки/фреймворки
3. Не поддерживаются реляционные базы данных
4. Может потребоваться дополнительная работа при миграции с GAE (если понадобится)
5. **Облачная платформа Amazon Web Services - AWS. Архитектура, регионы и зоны доступности.**

(AWS) — инфраструктура платформ облачных веб-сервисов, представленная компанией Amazon в начале 2006 года;

В инфраструктуре AWS представлено много сервисов для предоставления различных услуг, таких как:

1. хранение данных (файловый хостинг, распределённые хранилища данных),
2. аренда виртуальных серверов,
3. предоставление вычислительных мощностей и др.

Архитектура AWS:

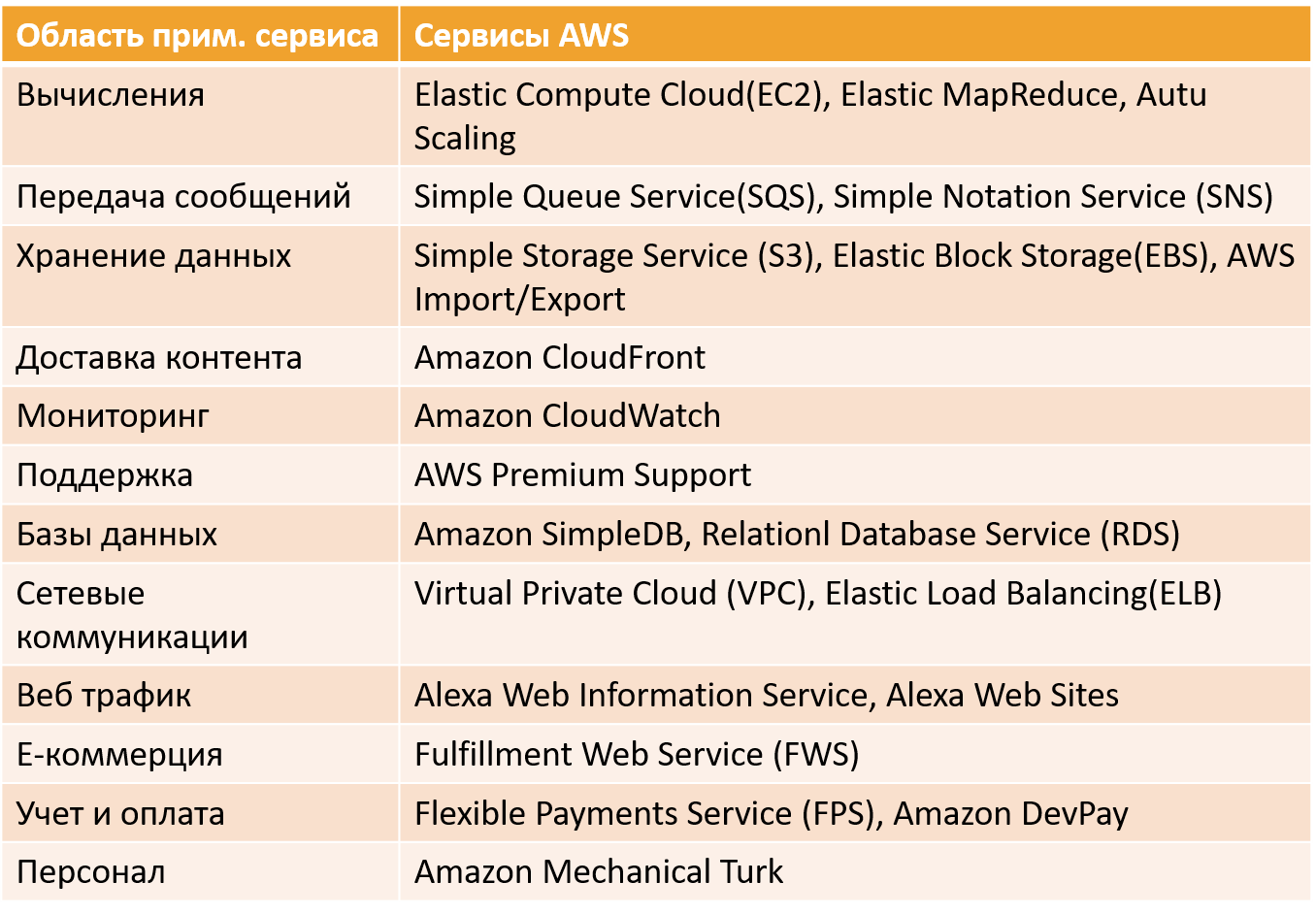
* EC2 – Elastic Compute Cloud;
* SQS – Simple QueueService;
* EBS – Elasyic block Service;
* S3 – Simple Storage Service;
* SNS – Simple Notification Service.

2020. Облако AWS охватывает **77 зон** доступности в **24 географических регионах** по всему миру (254 стран). В ближайшее время **планируется создать** еще **18 зон** доступности и **6 регионов** AWS

Зона доступности – это **один или несколько центров обработки данных** с резервным источником питания, сетевой конфигурацией и подключением в регионе AWS.

1. **Прикладные области и состав сервисов, публичной облачной системы AWS.**

Прикладные области сервисов AWS



1. **Способы управления AWS.**

Способы управления AWS:

* 1. Консоль управления
  2. Интерфейс CLI
  3. Пакеты SDK

В консоли собраны все сервисы AWS, но функциональность настройки несколько обрезана.

 В командной строке можно более гибко настроить тот или иной сервис, так же доступны закрытые в консоли функции

Пакеты SDK упрощают использование сервисов AWS в приложениях благодаря интерфейсу программирования приложений (API), адаптированному к используемому вами языку программированию или платформе

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

**Amazon Elastic Compute Cloud (EC2)** — это облачный сервис, предоставляющий:

* 1. виртуальные сервера
  2. 2 вида хранилищ данных:
     1. краткосрочное (исчезает вместе с ВМ)
     2. блочное (остается при удалении ВМ).
  3. балансировщик нагрузки.

Amazon EC2 Container Service (ECS) — это в высшей степени масштабируемый, высокопроизводительный сервис для управления контейнерами, который поддерживает контейнеры Docker.

Amazon Elastic Container Registry (ECR) – это полностью управляемый реестр контейнеров, который упрощает хранение, управление, публикацию и развертывание образов контейнеров и артефактов в любом месте

AWS Lambda позволяет запускать программные коды без выделения серверов и управления ими.

Auto Scaling помогает поддерживать доступность приложений и масштабировать ресурсы Amazon EC2 в обоих направлениях автоматически в соответствии с определенными вами условиями.

Функция Auto Scaling позволяет убедиться, что в системе работает нужное число экземпляров Amazon EC2.

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

Amazon Simple Storage Service (Amazon S3) — это хранилище объектов с простым интерфейсом веб-сервисов для хранения и извлечения любого объема данных практически из любого ресурса сети.

EBS (Elastic Block Storage) — это один из типов хранилища используемых в инстансах (виртуальных мащинах) EC2.

Amazon Elastic File System (Amazon EFS) представляет собой простое масштабируемое файловое хранилище, предназначенное для использования с инстансами Amazon EC2 в облаке AWS.

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

Amazon Relational Database Service— это сервис баз данных, который выносится на отдельную машину.

В Amazon RDS доступны следующие СУБД:

[**Amazon Aurora**](https://aws.amazon.com/rds/aurora/) **(**[MySQL](https://aws.amazon.com/rds/mysql/), [PostgreSQL](https://aws.amazon.com/rds/postgresql/))

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

[**Amazon Redshift**](https://aws.amazon.com/redshift/)

Тут очень много было инфы про бд, в пизду

Amazon SimpleDB – это высокодоступное хранилище данных [NoSQL](https://aws.amazon.com/ru/nosql/), которое полностью обеспечивает администрирование баз данных.

1. **AWS. Сервис Миграция, состав основных услуг.**
   1. AWS Application Discovery Service помогает системным интеграторам быстро и надежно планировать проекты миграции приложений, автоматически идентифицируя приложения, которые выполняются в локальных ЦОД, связанные с ними зависимости и их профили производительности.
   2. AWS Database Migration Service помогает легко и безопасно переносить базы данных в AWS. Исходная база данных во время миграции остается в полностью рабочем состоянии, что сводит к минимуму время простоя использующих ее приложений.
   3. AWS Server Migration Service — это безагентский сервис, упрощающий и ускоряющий **перенос тысяч локальных рабочих нагрузок в AWS**.
   4. AWS Snowball это **решение для переноса данных**, исчисляемых метаданными, предполагающее использование защищенных устройств для переноса крупных массивов данных в и из AWS.
   5. AWS Snowball Edge то **устройства переноса данных** объемом **100 ТБ** с —встроенными функциями хранения и вычисления.
   6. AWS Snowmobile это сервис для переноса данных, исчисляемых **эгсабайтами**, позволяющий переносить чрезвычайно крупные объемы данных в AWS.
2. **AWS. Сервис Сетевая конфигурация и доставка контента, состав основных услуг.**
   1. Amazon VPC C помощью Amazon Virtual Private Cloud (Amazon VPC) можно выделить **логически изолированный раздел** облака AWS, в котором можно запускать ресурсы AWS в определенной вами виртуальной сети.
   2. Amazon CloudFront Amazon CloudFront является сервисом глобальной сети доставки контента (**CDN**), ускоряющим доставку ваших веб-сайтов, API, видеоконтента или других сетевых ресурсов.
   3. Amazon Route 53 это облачный DNS сервис от Amazon.
   4. AWS Direct Connect Сервис AWS Direct Connect упрощает настройку выделенного сетевого соединения между корпоративной сетью и системой AWS.
   5. Elastic Load Balancing Elastic Load Balancing (ELB) автоматически распределяет входящий трафик приложения между несколькими инстансами EC2. Это обеспечивает более высокую отказоустойчивость приложений, поскольку необходимые для распределения трафика приложения ресурсы (ресурсы балансировки нагрузки) выделяются в нужном количестве.
3. **AWS. Передача сообщений. Очередь сообщений SQS Amazon.**

**Amazon Simple Queue Service** (Amazon SQS) — это быстрый, надежный, масштабируемый и полностью управляемый сервис для постановки сообщений в очередь.

Amazon SQS принимает сообщения и передает их серверам, подписанным на очередь сообщений.

Система обмена сообщениями позволяет многим компьютерам обмениваться информацией, не имея никаких сведений друг о друге.

Сервис принимает очереди сообщений для хранения.

При использовании Amazon SQS, разработчики могут просто переместить данные, распределённые между компонентами своих приложений, которые выполняют различные задачи, не теряя при этом сообщения.

При этом достигается высокая масштабируемость и надёжность.

Входит в инфраструктуру сервисов [Amazon Web Services](https://ru.wikipedia.org/wiki/Amazon_Web_Services).

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

Microsoft Azure — это открытая и гибкая облачная платформа, позволяющая быстро создавать, развертывать и управлять приложениями в глобальной сети центров обработки данных под управлением Майкрософт.

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

1. **Вычислительные службы Microsoft Azure.**

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

Azure предоставляет широкий выбор вариантов для размещения приложений и служб.

|  |  |
| --- | --- |
| **Имя службы** | **Функция службы** |
| Виртуальные машины Azure | Виртуальные машины Windows или Linux, размещенные в Azure. |
| Масштабируемые наборы виртуальных машин Azure | Масштабирование виртуальных машин Windows или Linux, размещенных в Azure. |
| Служба Azure Kubernetes | Управление кластерами для виртуальных машин с контейнерными службами. |
| Azure Service Fabric | Платформа распределенных систем, работающая в Azure или локальной среде. |
| Пакетная служба Azure | Управляемая служба для параллельных и высокопроизводительных вычислительных приложений. |
| Экземпляры контейнеров Azure | Контейнерные приложения выполняются в Azure без подготовки серверов или виртуальных машин. |
| Функции Azure | Управляемая событиями служба бессерверных вычислений. |

1. **Сетевые службы и Web приложения Microsoft Azure.**

|  |  |
| --- | --- |
| **Имя службы** | **Функция службы** |
| Виртуальная сеть Azure | Подключает виртуальные машины к входящим подключениям виртуальной частной сети (VPN). |
| Azure Load Balancer | Балансирует входящие и исходящие подключения к приложениям или конечным точкам служб. |
| Шлюз приложений Azure | Оптимизирует доставку фермы серверов приложений и повышает безопасность приложений. |
| VPN-шлюз Azure | Позволяет получить доступ к виртуальным сетям Azure через высокопроизводительные VPN-шлюзы. |
| Azure DNS | Обеспечивает сверхвысокую скорость ответов DNS и доступность домена. |
| Сеть доставки содержимого Azure | Доставляет содержимое с высокой пропускной способностью клиентам по всему миру. |
| Защита от атак DDoS Azure | Защищает приложения, размещенные в Azure, от распределенных атак типа "отказ в обслуживании" (DDoS). |
| Диспетчер трафика Azure | Распределяет сетевой трафик между регионами Azure по всему миру. |
| Azure Expres Route | Подключается к Azure через выделенные безопасные соединения с высокой пропускной способностью. |
| Наблюдатель за сетями Azure | Проводит мониторинг и диагностику неполадок в сети с помощью анализа на основе сценариев. |
| Брандмауэр Azure | Реализует высокий уровень безопасности и доступности брандмауэра с неограниченной масштабируемостью. |
| Виртуальная глобальная сеть Azure (WAN) | Создает единую глобальную сеть (WAN), подключенную к локальным и удаленным сайтам. |

1. **Службы базы данных Microsoft Azure**

|  |  |
| --- | --- |
| **Имя службы** | **Функция службы** |
| **Azure Cosmos DB** | Глобально распределенная база данных, которая поддерживает параметры NoSQL. |
| **База данных SQL Azure** | Полностью управляемая реляционная база данных с автоматическим масштабированием, интегральной аналитикой и надежной системой безопасности. |
| **База данных Azure для MySQL** | Полностью управляемая и масштабируемая реляционная база данных MySQL с высоким уровнем доступности и безопасности. |
| **База данных Azure для PostgreSQL** | Полностью управляемая и масштабируемая реляционная база данных PostgreSQL с высоким уровнем доступности и безопасности. |
| **SQL Server в виртуальных машинах Azure** | Служба, которая размещает корпоративные приложения SQL Server в облаке. |
| **Azure Synapse Analytics** | Полностью управляемое хранилище данных с интегрированной защитой для любых уровней масштабирования без дополнительных затрат. |
| **Azure Database Migration Service** | Служба, которая переносит базы данных в облако без изменения кода приложения. |
| **Кэш Redis для Azure** | Полностью управляемая служба, которая кэширует часто используемые и статические данные для снижения задержек в данных и приложениях. |
| **База данных Azure для MariaDB** | Полностью управляемая и масштабируемая реляционная база данных MariaDB с высоким уровнем доступности и безопасности. |

**80.     Сервисы SaaS Microsoft Azure.**

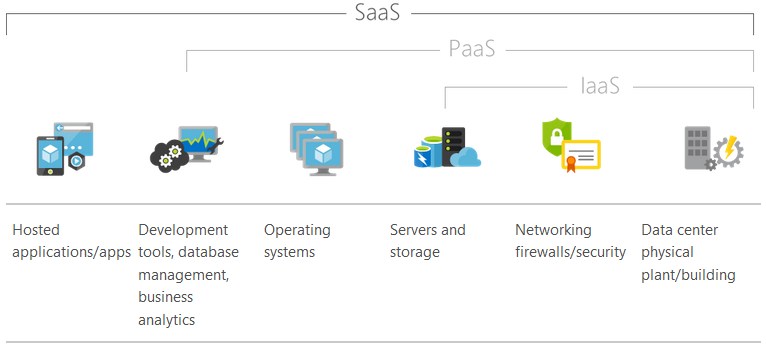
Программное обеспечение как услуга (SaaS) позволяет пользователям подключаться к облачным приложениям и использовать их через Интернет. Стандартными примерами могут быть:

* Windows Live Services. Office365, Microsoft 365,
* M-files – система управления контентом предприятия;
* Snelstat – популярное ПО для финансового бизнеса.
* Uber – По для связывания паасжира с водителем такси.

**SaaS** предоставляет полноценный набор программного обеспечения, который вы оплачиваете [поставщику облачных служб](https://azure.microsoft.com/ru-ru/overview/choosing-a-cloud-service-provider/) по мере использования. Вы арендуете использование приложения для вашей организации, и ваши пользователи подключаются к нему через Интернет, обычно с помощью веб-браузера.

**Преимущества SaaS**

* Доступ к продвинутым приложениям.
* Оплата только за те ресурсы, которые используете.
* Использование бесплатного клиентского программного обеспечения.
* Повышение мобильность ваших сотрудников.
* Доступ к данным приложений из любого места.



**81.     Служебная шина Microsoft Azure.  Управление обменом сообщений через сервисную шину в Microsoft Azure.**

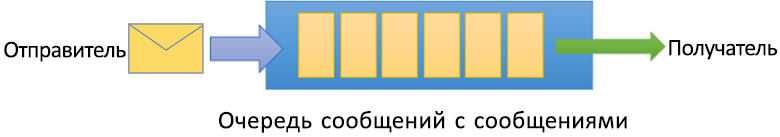
**Служебная шина Microsoft Azure** — это полностью управляемый брокер сообщений корпоративного типа с поддержкой очередей сообщений и разделов публикации и подписки.

* Поддерживает набор облачных технологий промежуточного уровня, ориентированных на обработку сообщений.
* Эти технологии представлены надежными очередями сообщений, а также возможностями публикации и подписки в рамках обмена сообщениями.
* Эти возможности обмена сообщениями через посредника могут рассматриваться как разделенные функции обмена сообщениями, поддерживающие публикацию и подписку, временное разделение, а также сценарии балансировки нагрузки с использованием рабочей нагрузки обмена сообщениями служебной шины.

Данные передаются между приложениями и (или) службами с помощью **сообщений**.

**Сообщением** называется контейнер с данными, дополненный метаданными. В нем могут содержаться любые данные, включая структурированные данные, закодированные в любом из распространенных форматов (например, JSON, XML, Apache Avro, обычный текст).

Сообщения отправляются и получаются через очереди. Очереди позволяют хранить сообщения, пока принимающее приложение сможет получить и обработать их.



* Очереди предлагают доставку сообщений конкурирующим потребителям по типу FIFO (первым пришел, первым вышел).
* Основное преимущество использования очередей — временное разделение компонентов приложений.
* Использование очередей в качестве посредника между производителями и потребителями сообщений уменьшает зависимость между компонентами. Так как производители и потребители не зависят друг от друга, обновление потребителя не оказывает влияния на производителя.

Создание Очереди:

* Выполнять операции управления обменом сообщениями служебной шины (очередей и разделов) можно с использованием класса [Microsoft.ServiceBus.NamespaceManager](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.servicebus.namespacemanager).
* Класс [NamespaceManager](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.servicebus.namespacemanager) предоставляет методы для создания, перечисления и удаления сущностей обмена сообщениями.
* Создав объект [Microsoft.ServiceBus.TokenProvider](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.servicebus.tokenprovider) на основе имени и ключа SAS, а также объект управления пространством имен службы, можно создать очередь, используя метод [Microsoft.ServiceBus.NamespaceManager.CreateQueue](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.servicebus.namespacemanager).

**82.     Обеспечения высокой доступности в Azure. Набор доступности. Домены отказа и обновлений.**

Все центры обработки данных Azure созданы для обеспечения высокой доступности. Резервные источники питания, несколько сетевых подключений и запасные массивы хранения данных — это лишь некоторые из основных концепции избыточности, которые предоставляет Azure и управляет ими.

Чтобы обеспечить высокую доступность приложения в Azure реализованы два метода гарантирующих такую доступность для рабочих нагрузок IaaS:

* + Набор доступности;
  + Зона доступности.

**Набор доступности** (***availability set***) – это метод логического группирования виртуальных машин, работающих в одной зоне, путем распределения их по одной в стойках центра обработки данных Azure.

В качестве логических групп используются:

* + Домены **отказов**;
  + Домены **обновлений**.

В результате чего обеспечивается **минимизация сбоев приложений**, работающих на этих ВМ из-за отказов или проведения технического обслуживания с целью обновления ПО или конфигурация ВМ.

Azure по требованию пользователя может распределять виртуальные машины на оборудовании физических серверов ЦОД для удовлетворения **требований к избыточности**.

**Домен отказов** - это логическая группа оборудования в центре обработки данных Azure.

Формирование домена отказов – это способ резервирование аппаратных средств на случай различных видов аварийных ситуаций, которые могут возникнуть в процессе работы ЦОД:

* + Выход из строя аппаратных средств серверов;
  + Отказы в системах электропитания и охлаждения;
  + Отказы в работе сетевого оборудования;
  + Отказы в работе систем хранения на которых располагаются управляемые диски ВМ.

Домен отказов содержит оборудование с общим питанием или сетевым оборудованием. Платформа Azure логически назначает кластеры хранения доменам сбоя, чтобы гарантировать, что управляемые диски виртуальных машин, распределенных по доменам сбоев, также распределяются по соответствующим кластерам хранения. Azure сам формирует эти домены и отслеживает процесс распределения вновь создаваемых виртуальных машин по доменам отказа.

**Домен обновлений**

* Помимо аппаратных отказов, причиной неработоспособности ВМ может быть проведение операций по обновлению ПО, установке патчей ОС или проведение иных работ связанных с проведением планового обслуживания программных и аппаратных средств ЦОД.
* Для **защиты от простоев** в этих случаях в Azure формируются соответствующие логические группы - **домены обновлений**.
* Такой порядок проведения обслуживания применяется для того, **чтобы не все виртуальные машины**, входящие в одну **группу доступности** были **остановлены одновременно** при выполнении **технического обслуживания**.

**83.     Обеспечения высокой доступности в Azure. Зона доступности. Платформа высокой доступности.**

Все центры обработки данных Azure созданы для обеспечения высокой доступности. Резервные источники питания, несколько сетевых подключений и запасные массивы хранения данных — это лишь некоторые из основных концепции избыточности, которые предоставляет Azure и управляет ими.

Чтобы обеспечить высокую доступность приложения в Azure реализованы два метода гарантирующих такую доступность для рабочих нагрузок IaaS:

* + Набор доступности;
  + Зона доступности.

**Зоны доступности (availability zone)** - это физически отдельные центры обработки данных, которые работают на независимых системах жизнеобеспечения, таких как электропитание, охлаждение и сетевое оборудование подключения к сети.

* Каждый регион Azure, поддерживающий Зоны доступности предоставляет **три зоны**.
* Виртуальные сетевые ресурсы Azure, такие как общедоступные IP-адреса и балансировщики нагрузки, могут охватывать все зоны в регионе, чтобы обеспечить избыточность не только для виртуальных машин, но и для сетевого оборудования.
* При развертывании больших приложений можно создать более одной виртуальной машины в каждой доступности. зона.
* Несколько виртуальных машин в зоне доступности не совсем соответствуют методике распределение ВМ по доменам обновлений и сбоев, как имеет место в группах доступности

**Организация сетевых ресурсов в зонах доступности**

* Когда сетевые ресурсы подключены к одному центру обработки данных Azure, или зоне то, сбой в этом ЦОД приводит к тому, что все приложение становится недоступным для клиент.
* Не имеет значения, что другие виртуальные машины продолжают работать в других зонах. Без подключения к сети для распределения трафика от ваших клиентов весь приложение недоступно.
* Общедоступный IP-адрес, охватывающий зоны доступности, обеспечивает единую точку входа для клиентов, чтобы связаться с вашим балансировщиком нагрузки, а затем распределить поток запросов между доступными виртуальными машинами.

|  |  |  |  |
| --- | --- | --- | --- |
| **Северная и Южная Америка** | **Европа** | **Африка** | **Азиатско-Тихоокеанский регион** |
| Центральная Канада | Центральная Франция | Южная Африка, Северный \* | Japan East |
| Центральная часть США | Центрально-Западная Германия |  | Southeast Asia |
| Восточная часть США | Северная Европа |  | Восточная Австралия |
| восточная часть США 2 | южная часть Соединенного Королевства |  |  |
| Юго-Центральный регион США | Западная Европа |  |  |
| US Gov (Вирджиния) |  |  |  |
| Западная часть США 2 |  |  |  |

**Регионы Azure поддерживаающие зоны доступности**

**Платформы высокой доступности**

В последнее время Azure представила множество новых функций в отношении высоких доступность для PaaS. Один из них связан с контейнерами и экосистемой окружающие их. В Azure представлены следующие службы:

* 1. Контейнеры в сервисах приложений
  2. Группы экземпляров контейнера Azure
  3. Службы Azure Kubernetes
  4. Другие оркестраторы контейнеров, такие как DC/OS и Swarm.

Другой важной платформой, обеспечивающей высокую доступность, является Service Fabric.

* Обе Service Fabric и оркестраторы контейнеров, включающие Kubernetes, гарантируют, что желаемое количество экземпляров приложения всегда запущено и работает в среда. Это означает, что даже если один из экземпляров выйдет из строя окружение, оркестратор узнает об этом посредством активного мониторинга и будет запускать новый экземпляр на другом узле, тем самым поддерживая идеальный количество экземпляров.

**84.     Балансировка нагрузки. Структура балансировщика нагрузки на примере Azure.**

* Балансировка нагрузки это способ горизонтального масштабирования приложений в Azure.
* Как и обеспечение высокой доступности (готовности) он  основан на использовании избыточности ресурсов в случае увеличения масштаба потока запросов к облачной службе (приложению).
* Балансировщики нагрузки - это сетевые ресурсы, которые принимают входящий трафик  от клиентов приложений и распределяют его с помощью применения фильтров и правил балансировки нагрузки.
* Балансировщики нагрузки в Azure могут работать на двух разных уровнях:
* уровень 4 (транспортный уровень стека ВОС),
* уровень 7 (прикладной уровень стека ВОС)
* Оба уровня балансировщика нагрузки работают одинаковым способом.

Состав балансировщика нагрузки:

* Пул IP-адресов внешнего интерфейса - точка входа в подсистему балансировки нагрузки. Чтобы разрешить доступ из Интернет, общедоступный IP-адрес может быть присоединен к пулу внешних IP-адресов.
* Зонды работоспособности - отслеживают состояние подключенных виртуальных машин. Чтобы убедиться, что трафик распространяется только на исправные и быстро реагирующие виртуальные машины, регулярно выполняются проверки состояния ВМ на основе контроля того, что виртуальная машина правильно реагирует на трафик.
* Правила балансировщика нагрузки: определяют порядок распределения трафика по виртуальным машинам. Каждый входящий пакет сравнивается с правилами, которые определяют входящие протоколы и порты, а затем распределяют его  по виртуальным машинам. Если для входящего трафика нет правил, то этот трафик отбрасывается.
* Правила трансляции сетевых адресов (NAT) – эти могут направлять трафик напрямую к определенным ВМ. Например, если вы хотите предоставить удаленный доступ через SSH или RDP, вы может определять правила NAT для перенаправления трафика с внешнего порта на одну из виртуальных машин.
* Внутренний пул IP-адресов – это внутренние адреса виртуальных машин, на которых выполняется приложение.

Балансировщики нагрузки могут работать в одном из двух разных режимов:

* Балансировщик нагрузки Интернета.
* Внутренний балансировщик нагрузки

**85.     Push и Pull модели распределения нагрузки. Способы балансировки нагрузки.**

Чтобы распределить рабочую нагрузку между несколькими узлами можно выбрать модель push или pull.

* С моделью **push** вы можете использовать эластичную балансировку нагрузки (ELB) для распределения рабочей нагрузки.
* ELB направляет входящие запросы приложений распределяя их между несколькими экземплярами EC2.
* При  распределении трафика балансировщик сетевой нагрузки работает на уровне 4 модели взаимодействия открытых систем (OSI) обеспечивая скорость обработки на уровне миллионов запросов в секунду.
* Для асинхронных, управляемых событиями рабочих нагрузок можно реализовать **pull** модель
* В модели pull задачи, которые необходимо выполнить, или данные, которые необходимо обработать, могут храниться в виде сообщений в очереди с помощью Amazon Simple Queue Service (Amazon SQS) или в виде решения для потоковой передачи данных, например Amazon Kinesis.
* Затем несколько вычислительных ресурсов могут извлекать и использовать эти сообщения, обрабатывая их распределенным образом.

Способы:

* Балансировка нагрузки на домене. Самый простой способ — балансировать запросы клиентов через ваше доменное имя. Можно попробовать на уровне DNS-сервера добавить к домену второй адрес. Это приведет к тому, что часть запросов на домен попадет на наш первый адрес, а часть — на второй.
* Балансировщик. Вариант номер два — на ваш домен мы ставим специальный сервер, который умеет принимать запросы от клиентов, распределять и передавать их дальше. Это простая и быстрая операция, она дает возможность отвечать клиентам почти мгновенно.
* Комбинация 2-х видов балансировки

**86.     Причины появления культуры DevOps. Цели и задачи DevOps.**

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

За последние десять лет или около того было разработано множество инструментов и методов, которые в конечном итоге позволил сформировать культуру DevOps.

Все эти методы и инструменты сводятся к одному:

* обеспечение постоянства устойчивого развертывания версий ПО в условиях несогласованности сред развертывания.

Причины возникновения DevOps:

* возрастающее значение концепции PaaS.
* появление новых систем, для которых работа с данными и предоставление различного рода информационных услуг является основной функцией, это своего рода фабрики информации.

Цели и задачи DevOps:

Поскольку процессы девопс охватывают весь цикл поставки ПО, выделяют несколько основных целей этого подхода:

* сокращение времени для выхода на рынок;
* снижение частоты отказов новых релизов;
* сокращение времени выполнения исправлений;
* уменьшение количества времени на восстановления при сбое новой версии или других случаях отключения текущей системы.

Цели DevOps достигаются через решение следующих задач:

* согласование процессов разработки и поставки ПО с эксплуатацией;
* автоматизация процессов разработки, тестирования и развертывания;
* непрерывное тестирование качества приложений;
* управление ИТ-инфраструктурой как кодом;
* управление изменениями;

**87.     DevOps. Инфраструктура как код.**

Идея IaC (Infrastructure as Code) состоит в том, чтобы явно зафиксировать все требования к инфраструктуре в виде недвусмысленного кода на некотором формальном языке:

YAML, JSON, HCL

Язык DSL (Chef, Puppet, Ansible и т.д.)

Ruby, Python, Go, PowerShell, …

Код описания конфигурации не подлежит интерпретации человеком и может точно выполняться компьютерами, в результате обеспечивается согласованность конфигураций разных сред развертывания ПО.

Сегодня IaC это основа облачных вычислений и неотъемлемая часть DevOps.

Инфраструктурный репозиторий включает:

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

Разрешение конфликтов между разными видами ПО инфраструктуры  и различными версиями разрертываемого ПО:

* Эта проблема возникает, когда необходимо развернуть ПО на серверах:
* имеющих разные версии ОС, системных библиотек, сервисов и сетевых служб;
* На которых уже установлены другие версии разрабатываемого ПО.
* Возможные решения:
* Ручная установка «заново» системы начиная с установки ОС, ПО стандартных служб и кончая установкой созданного ПО.
* Использование ПО для  клонирования дисков, например Ghost, Acronix и т.п.
* Недостатки:
* Большая длительность процесса развертывания.
* Современное решение: - изоляция вариантов развертывания друг от друга.

**88.     DevOps. Конвейер непрерывного развертывания как код.**

Jenkins — это сервер непрерывной интеграции, написанный на Java.

Он является чрезвычайно расширяемой системой из-за внушительной экосистемы разнообразных плагинов.

Настройка пайплайна осуществляется в декларативном или императивном стиле на языке Groovy, а сам файл конфигурации (Jenkinsfile) располагается в системе контроля версий вместе с исходным кодом.

Это удобно для небольших проектов, однако, часто более практично хранить конфигурации всех сервисов в отдельном репозитории.

Jenkins Pipeline позволяет в виде кода определить весь жизненный цикл приложения.

Jenkins Pipeline - это комбинация плагинов, которые поддерживают интеграцию и реализацию конвейеров непрерывной доставки с использованием Jenkins.

**Конвейер как код** описывает набор функций, которые позволяют пользователям Jenkins определять конвейерные процессы заданий с кодом, который хранится и контролируется версиями в исходном репозитории. …

Чтобы использовать конвейер в качестве кода, проекты должны содержать файл с именем **Jenkinsfile** в корне репозитория, который содержит «сценарий конвейера».

**89. Реализация DevOps в AWS.**

**AWS CodeCommit** — это полностью управляемая служба контроля исходного кода, реализация GIT, полностью его повторяющая. Сам код храниться на S3.

**AWS CodeBuild** - сборочный сервер — для проектов, требующих **сборки** перед развёртыванием, например, Java. По умолчанию запускает контейнер на базе Ubuntu, но можно указать и свой собственный. Может быть использован в качестве тестового.

**AWS CodeDeploy - с**ервис для развёртывания кода, который с помощью предустановленного агента и гибких настроек работает в любом окружении. Особенным отличием является то, что агент работает не только с Amazon AWS виртуалками, но и «внешними», что позволяет централизованно разворачивать самое разношерстное ПО, в т.ч. и локально.

**AWS CodePipeline –** конвейер развертывания, взаимодействует с предыдущими тремя сервисами, запуская их в нужной последовательности и, собственно, обеспечивая автоматизацию DevOps-процессов.

**AWS CodeStar** - сервис, по сути дублирующий **CodePipeline**, но ориентрированный на простоту запуска и настройки, что решается с помощью широкого **набора готовых шаблонов** (под связку приложение-язык) и действительно удобного Dashboard с плагинами, имеющими некоторую интеграцию с сервисом мониторинга (**CloudWatch**) и плагином для интеграции с Jira.

AWS CodeCommit — это полностью управляемая служба контроля исходного кода, упрощающая размещение защищенных и в высшей степени масштабируемых частных репозиториев Git.

AWS CodeCommit устраняет необходимость в использовании собственной системы контроля исходного кода.

AWS CodeCommit – это полностью управляемый сервис по управлению исходным кодом, предназначенный для хостинга защищенных репозиториев на основе Git.

Он упрощает совместную работу команд с кодом в безопасной системе с высокой масштабируемостью. С сервисом CodeCommit не требуется поддерживать собственную систему управления исходным кодом или беспокоиться о масштабировании соответствующей инфраструктуры.

С помощью CodeCommit можно безопасно хранить любые ресурсы, от исходного кода до исполняемых файлов.

Сервис эффективно интегрируется с существующими инструментами Git.

AWS CodeBuild — это полностью управляемый сервис сборок, который **компилирует** исходный код, **запускает тесты** и создает готовые к развертыванию **пакеты** программного обеспечения.

Благодаря CodeBuild больше не нужно распределять и масштабировать собственные серверы сборок, а также обеспечивать управление ими.

 CodeBuild непрерывно масштабируется и способен обрабатывать **несколько сборок одновременно**, поэтому сборки не будут ждать в очереди.

AWS CodeBuild – это полностью управляемый сервис:

создания сборок,

выполняющий компиляцию исходного кода,

проводящий тестирование и

формирующий готовые к развертыванию пакеты программного обеспечения.

AWS CodeDeploy — это сервис **автоматизации развертываний** кода в любом инстансе, включая инстансы EC2 и инстансы, выполняемые локально.

AWS CodeDeploy упрощает быстрый выпуск новых возможностей, помогает избежать простоев во время развертывания и **упрощает весь процесс обновления** приложений.

AWS CodeDeploy можно использовать для автоматизации развертываний ПО, избавляясь от необходимости выполнять операции вручную с риском совершения ошибки.

Этот сервис **масштабируется** вместе с вашей инфраструктурой, так что вы легко сможете выполнить развертывание в одном или **тысячах** инстансов.

Cервис для **автоматизации развертывания программного обеспечения** с использованием различных вычислительных сервисов, таких как:

Amazon EC2,

AWS Fargate,

AWS Lambda и

локальных серверов.

AWS CodeDeploy упрощает быстрый выпуск новых возможностей, помогает избежать простоев во время развертывания и берет на себя все сложности при обновлении приложений.

AWS CodeDeploy можно использовать для автоматизации развертывания программного обеспечения, устраняя необходимость в подверженных ошибкам ручных операциях.

Сервис масштабируется в соответствии с вашими потребностями.

AWS CodePipeline — это сервис **непрерывной интеграции и доставки**, позволяющий быстро и надежно выполнять обновления приложений и инфраструктуры.

CodePipeline выполняет **компоновку**, **тестирование** и **развертывание** вашего кода каждый раз **при внесении в него изменений**, используя определяемые вами модели процесса выпуска.

Это позволяет быстро и надежно осуществлять доставку различных возможностей и обновлений.

Это центральный инструмент для организации DevOps на Amazon AWS.

Взаимодействует с сервисами:

CodeCommit;

CodeBuild;

CodeDeploy.

запуская их в нужной последовательности и, собственно, **обеспечивая автоматизацию DevOps-процессов**

Позволяет;

организовывать ветвления процессов,

запускать сторонние сервисы (например, для тестирования),

делать параллельные ветки,

запрашивать подтверждение (*Approval Actions*) перед запуском следующего этапа.

Сервис, по сути дублирующий **CodePipeline.**

С помощью AWS CodeStar можно:

настроить весь набор инструментов для непрерывной доставки ПО за считаные минуты, чтобы быстрее начать выпуск кода.

AWS CodeStar облегчает безопасную совместную работу команды.

С помощью шаблонов проекта, предлагаемых AWS CodeStar, можно легко разрабатывать разнообразные приложения:

от веб-сайтов и интернет-приложений

до интерактивных сервисов и приложений с использованием технологий Amazon Alexa.

Шаблоны проектов AWS CodeStar включают необходимый для начала работы код на поддерживаемых языках программирования, включая Java, JavaScript, PHP, Ruby, C# и Python.

Сервисы Amazon AWS, реализующие концепцию «**Инфраструктура как код**», это:

**Elastic Beanstalk**

**OpsWorks**

**CloudFormation**

Чтобы сделать что-то быстро (и это не значит, что плохо), это какой-то стандартный функционал (например, простой сайт) — удобно использовать **Elastic Beanstalk**.

Если это сложный проект с многочисленными вложенными элементами и серьёзными требованиям по сетевым настройкам — без использования **CloudFormation** не обойтись.

Как нечто среднее — представляется использование **OpsWorks**, базирующегося на Chef.

На сложных проектах обычно используется комбинация из всех трёх служб IaaC:

**CloudFormation** :

поднимает основную инфраструктуру (**VPC**, подсети, репозитории, создаёт нужные роли в **IAM** для доступа и т.д.),

после запускает **OpsWorks**-стек, который уже может гибко настроить внутреннюю составляющую запущенных виртуалок.

может поднять и стек для **Elastic Beanstalk** компонентов, чтобы разработчики с помощью расширений могли сами менять некоторые параметры работающего приложения (количество и тип используемых виртуалок, использование *Load Balancer* и т.д.) просто изменяя простой файл конфигурации в папке с кодом, когда применение изменений (в т.ч. в инфраструктуру приложения) происходит автоматически после коммита.

Jenkins - это сервер автоматизации с открытым исходным кодом, который интегрируется с рядом сервисов AWS, таких как AWS CodeCommit, AWS CodeDeploy, Amazon EC2 Spot и Amazon EC2 Fleet.

Вы можете использовать Amazon Elastic Compute Cloud (Amazon EC2) для развертывания приложения Jenkins на AWS за считанные минуты.

Функционирующий конвейер создает полностью управляемую службу сборки, которая компилирует исходный код.

Затем он создает артефакты кода, которые CodeDeploy может использовать для автоматического развертывания в производственной среде.

Рабочий процесс развертывания начинается с размещения кода приложения в репозитории GitHub.

Чтобы автоматизировать этот сценарий, добавлено управление исходным кодом в проект Jenkins в разделе «Исходный код».

Выбран вариант GitHub, который по дизайну клонирует копию содержимого репозитория GitHub в локальном каталоге рабочей области Jenkins.

На втором этапе процедуры автоматизации включается триггер для сервера Jenkins, используя параметр «Опрос SCM». Эта опция заставляет Дженкинс проверять настроенный репозиторий на предмет любых новых коммитов/изменений кода с указанной частотой.

AWS X-Ray помогает разработчикам **анализировать и отлаживать** распределенные приложения **в рабочей среде** и на этапе разработки, например приложения, созданные с использованием архитектуры микросервисов.

С помощью X-Ray можно **оценить производительность** своего приложения и базовых сервисов, чтобы найти и устранить основную причину проблем производительности и ошибок.

X-Ray обеспечивает **комплексное отслеживание запросов** и их маршрутов в приложении и показывает карту внутренних компонентов приложения.

X-Ray используется для **анализа приложений на стадии разработки и развертывания**, от простых **трехступенчатых** вариантов до сложных приложений, состоящих из **тысяч микросервисов**.

**90. Реализация DevOPs в Azure.**

В Azure DevOps входят:

**Azure Pipelines  -** Непрерывная интеграция и развертывание, поддержка любых языков, платформ и облачных сервисов. Подключение к GitHub и любому репозиторию Git, непрерывное развертывание.

**Azure Boards -** Мощные инструменты контроля рабочего процесса: канбан-доски, журналы невыполненных работ, командные информационные панели и настраиваемые отчеты.

**Azure Artifacts -** Каналы пакетов Maven, npm и NuGet из открытых и закрытых источников.

**Azure Repos -** Закрытые облачные репозитории Git неограниченного объема для хранения файлов проекта Совместные запросы на включение, улучшенное управление файлами и многое другое.

**Azure Test Plans -** Комплексное решение для планирования и произвольного тестирования.

Для внедрения непрерывной интеграции и непрерывной доставки (CI/CD) для проектов программного обеспечения можно разместить развертывание Jenkins в Azure либо расширить имеющуюся конфигурацию Jenkins с помощью ресурсов Azure.

Кроме того, можно использовать подключаемые модули Jenkins, чтобы упростить непрерывную интеграцию и непрерывную доставку приложений в Azure.

Azure DevOps – это результат развития Visual Studio Team Services (VSTS). Пользователи VSTS будут автоматически переведены на Azure DevOps. Существующие пользователи не потеряют в функциональности, однако приобретут больше возможностей выбора и контроля. Полная прозрачность и интеграция, отличительные особенности VSTS, сохранились. Службы Azure DevOps отлично сочетаются друг с другом.

Перевод пользователей на использование Azure DevOps включает:

URL-адреса изменятся с abc.visualstudio.com на dev.azure.com/abc.

Будет включена переадресация с адреса visualstudio.com, поэтому вы не столкнетесь с битыми ссылками.

Поскольку служба изменилась, обновился и интерфейс. У новых пользователей он будет включен по умолчанию. Он станет основным и для существующих пользователей.

Локальный сервис Team Foundation Server (TFS) будет по-прежнему обновляться, обновления будут аналогичны тем, что выпускаются для Azure DevOps.

После выхода следующей версии TFS продукт получит новое название — Azure DevOps Server и будет обновляться в прежнем режиме.

**91.  Проблемы переноса приложений в «облако».**

Итак, одна из основных проблем – это перенос существующих приложений в “облако”. Во-первых, это не всегда возможно сделать из-за особенностей архитектуры конкретного приложения, его привязок к другим системам или сервисам, которые ещё/уже не будут перенесены в “облака”. Зачастую переход к облачным вычислениям невозможен из-за использования специфических API ОС или вызова низкоуровневых функций для оптимизации работы. Таких приложений, может быть, не очень много, но они есть. Бывают и такие случаи, когда перенос теоретически возможен, но это требует либо значительной переработки кода, либо переписывания всего набора ПО с нуля. Очень часто это экономически невыгодно

Ошибки и подводные камни миграции

Нередко при миграции в облако компании допускают ошибки, а все потому, что не уделяют должного внимания некоторым важным деталям. Но если соблюдать установленные рекомендации и действовать по плану, конечный результат будет оправданным и ожидаемым.

Ошибка 1. Отсутствие схемы зависимости приложений

При миграции в облако следует уделить внимание схеме зависимости приложений друг от друга и инфраструктуры в целом. Результаты лучше всего отражать визуально в виде карты зависимостей. Допустим, приложения A, B, C используют одну и ту же базу данных. Следовательно, необходимо учесть это и проработать механизм комбинированного перемещения в облако. Если этого не сделать, велика вероятность некорректной работы приложения.

Ошибка 2. Отсутствие плана миграции

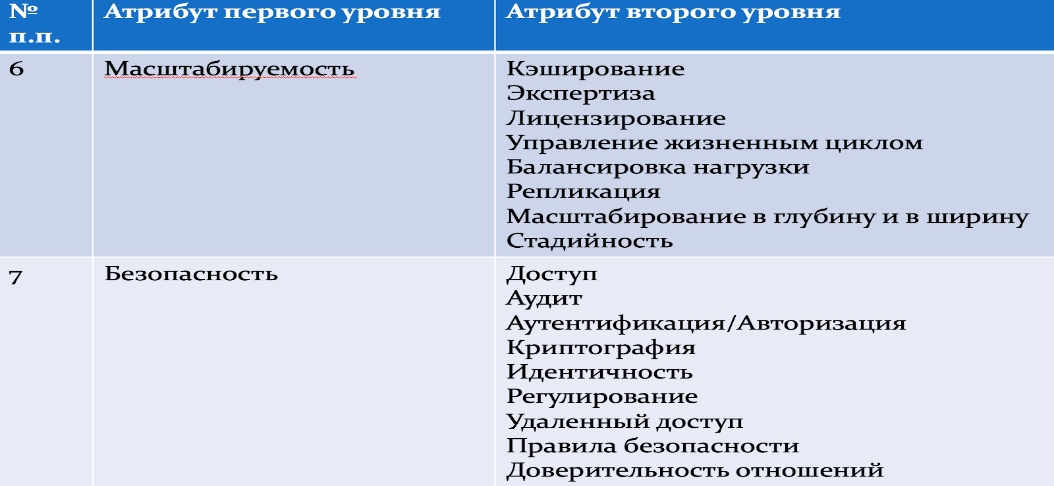
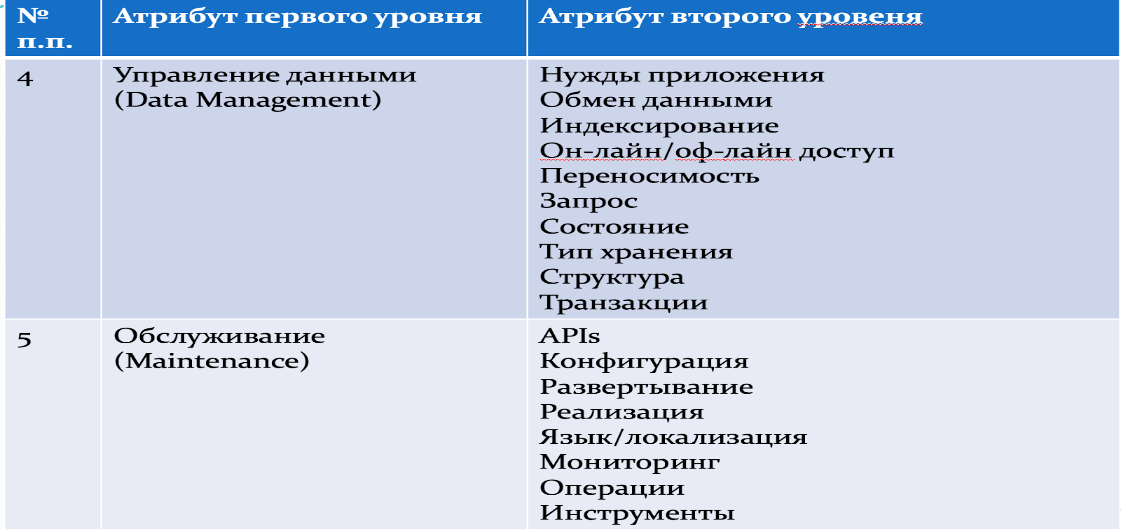
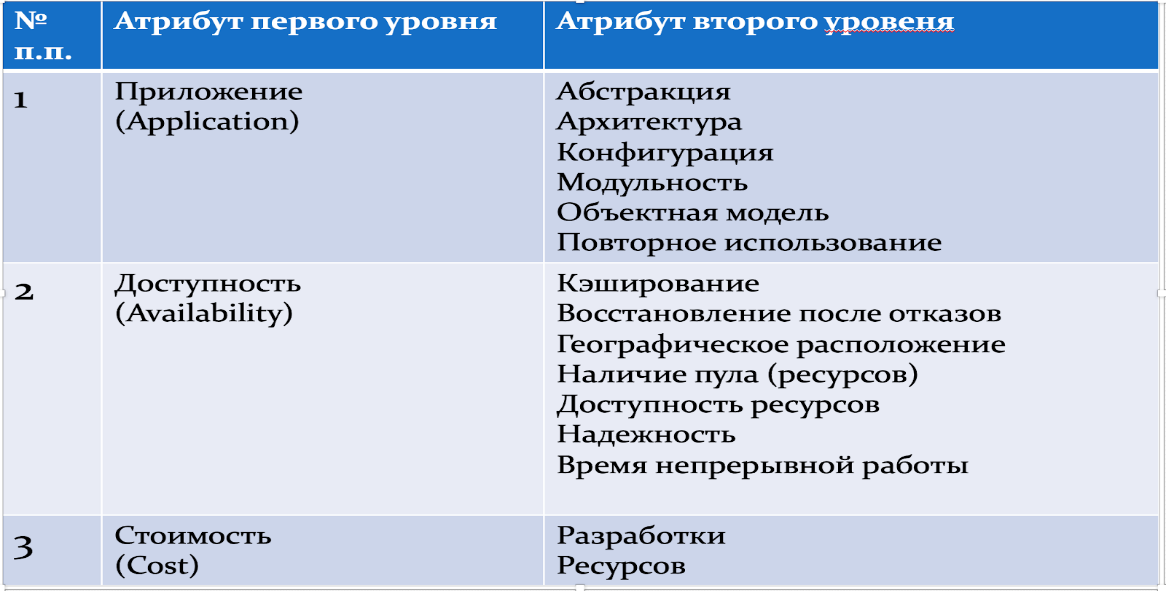
Что вы будете переносить в облако, в какой последовательности, в какой момент, в какие сроки? До начала миграции подготовьте ответы на эти и другие вопросы, отобразив основные шаги в плане миграции. План позволяет прописать каждый этап переезда, избавляя от хаотичных действий и необдуманных шагов. Если в инфраструктуре компании существуют приложения, перенос которых невозможно разделить на части, следует придерживаться варианта единовременного, неделимого переноса и максимального контроля каждого выполненного шага. Анализ ситуации и выявление ошибок на ранних этапах помогут сэкономить время и добиться желаемых результатов.

Ошибка 3. Запуск миграции без предварительных тестов

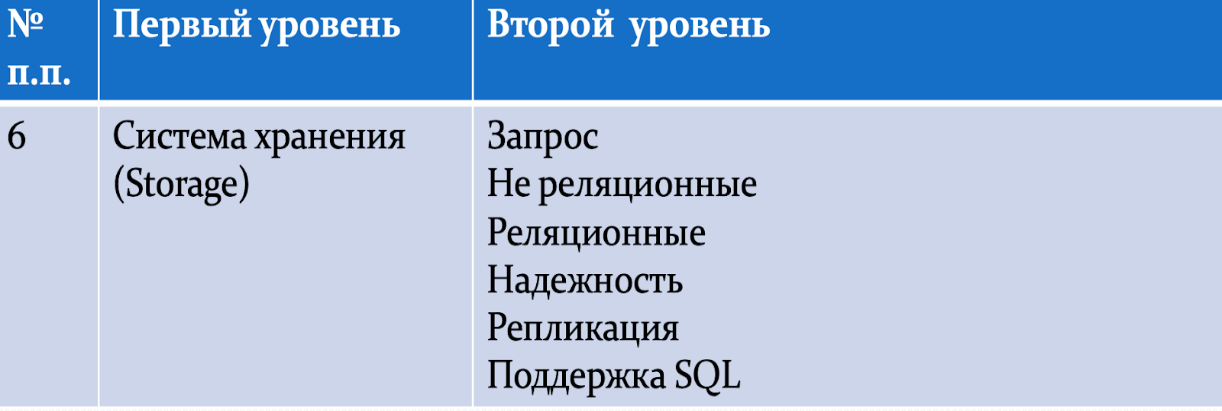
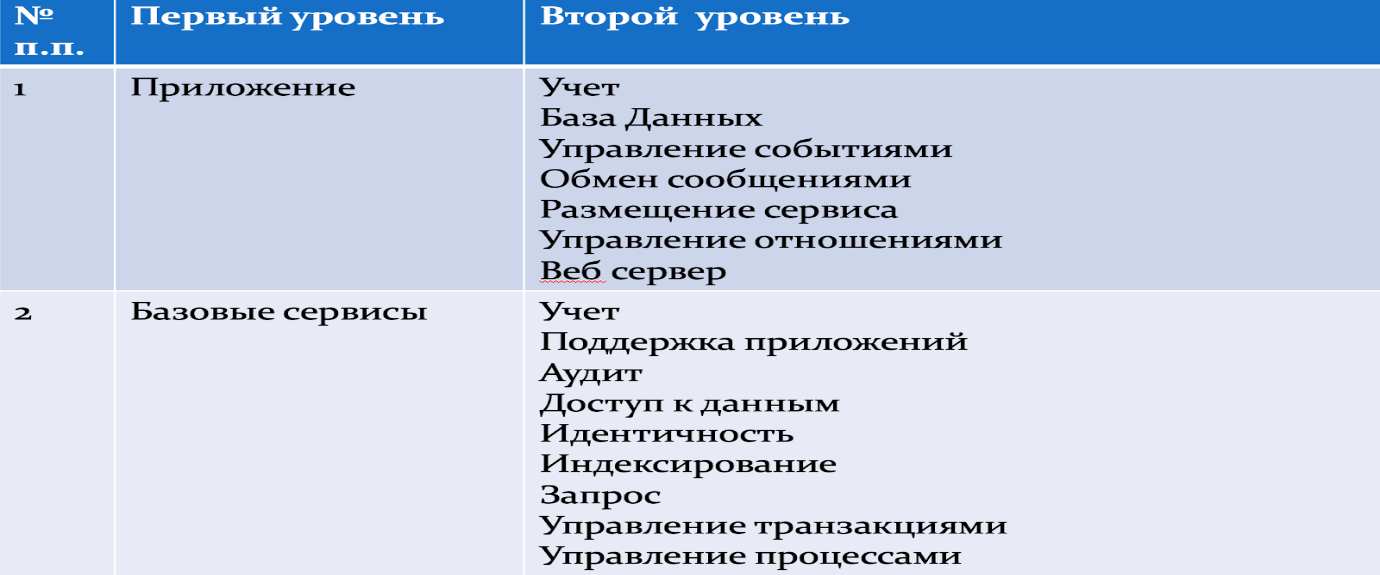
Прежде чем мигрировать в облако, выберите надежного и проверенного хостинг-провайдера. В одной из статей мы уже рассказывали, как это делать. Выбрав поставщика, запросите тестовый доступ к облачной площадке и проработайте процессы миграции. Сначала перенесите в облако «легкий» сервис, оцените потраченное время, проверьте, как все работает, проанализируйте, нет ли ошибок, и перейдите к следующему сервису. Последовательно выполненные мероприятия по переносу инфраструктуры в дальнейшем дадут хорошие результаты.

**92.  Функциональный маппинг облачных приложений и облачных сервисов.**

**93.  Атрибуты (характеристики) приложений переносимых в облако.**



**94.  Атрибуты (характеристики) облачных сервисов.**



**95. Обзор существующих угроз информационной безопасности облачных вычислений.**

1. Утечка данных

Из-за большого количества данных, которые сегодня часто переносятся в облака, площадки облачных хостинг-провайдеров становятся привлекательной целью для злоумышленников. При этом серьезность потенциальных угроз напрямую зависит от важности и значимости хранимых данных.

**Хули делать?????**

CSA рекомендует использовать многофакторную аутентификацию и шифрование.

1. **Компрометация учетных записей и обход аутентификации**

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

**Хули делать?????**

**CSA рекомендует использовать механизмы многофакторной аутентификации, включая одноразовые пароли, токены, смарт-карты, USB-ключи**

1. **Взлом интерфейсов и API**

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

**Хули делать?????**

**CSA рекомендует:**

**-организовать адекватный контроль доступа,**

**- использовать инструменты защиты и раннего обнаружения угроз.**

**- умение моделировать угрозы и находить решения по их отражению — достойная профилактика от взломов. - выполнять проверку безопасности кода и запускать тесты на проникновение.**

1. **Уязвимость используемых систем**

**Уязвимость используемых систем — проблема, встречающаяся в мультиарендных облачных средах.**

**Распространена ошибка, когда при использовании облачных решений в модели IaaS компании уделяют недостаточно внимания безопасности своих приложений, которые размещены в защищенной инфраструктуре облачного провайдера.**

**И уязвимость самих приложений становится узким местом в безопасности корпоративной инфраструктуры.**

**Хули делать?????**

**-регулярное сканирование на выявление уязвимостей,**

**-применение последних патчей ;**

**-быструю реакцию на сообщения об угрозах безопасности.**

1. **Кража учетных записей**

**Фишинг, мошенничество, эксплойты встречаются и в облачном окружении.**

**Сюда добавляются угрозы в виде попыток манипулировать транзакциями и изменять данные.**

**Хули делать?????**

**-Необходимо запретить «шаринг» учетных записей пользователей и служб между собой, а также обратить внимание на механизмы многофакторной аутентификации.**

**- Сервисные аккаунты и учетные записи пользователей необходимо контролировать, детально отслеживая выполняемые транзакции.**

**- Главное — обеспечить защиту учетных записей от кражи, рекомендует CSA.**

1. **Инсайдеры-злоумышленники**

**Инсайдерская угроза может исходить от нынешних или бывших сотрудников, системных администраторов, подрядчиков или партнеров по бизнесу.**

** Инсайдеры-злоумышленники преследуют разные цели, начиная от кражи данных до желания просто отомстить.**

** В случае с облаком цель может заключаться в полном или частичном разрушении инфраструктуры, получении доступа к данным и прочем.**

**Системы, напрямую зависящие от средств безопасности облачного поставщика, — большой риск.**

**Хули делать?????**

** Позаботиться о механизмах шифрования и взять под собственный контроль управление ключами шифрования.**

**Не стоит забывать про логирование, мониторинг и аудит событий по отдельно взятым учетным записям.**

1. **Целевые кибератаки**

**Злоумышленника, задавшегося целью установить и закрепить собственное присутствие в целевой инфраструктуре, не так легко обнаружить. (Развитая устойчивая угроза, или целевая кибератака)**

**Для минимизации рисков и профилактики подобных угроз поставщики облачных услуг используют продвинутые средства безопасности.**

**Но помимо современных решений, требуется понимание сущности и природы такого вида атак.**

**Хули делать?????**

**-проводить специализированное обучение сотрудников по -распознаванию техник злоумышленника,**

**-использовать расширенные инструменты безопасности,**

**-уметь правильно управлять процессами,**

**-знать о плановых ответных действиях на инциденты,**

**-применять профилактические методы,**

**-повышающие уровень безопасности инфраструктуры.**

1. **Перманентная потеря данных**

**Облачные хостинг-провайдеры для соблюдения мер безопасности рекомендуют отделять пользовательские данные от данных приложений, сохраняя их в различных локациях.**

**Не стоит забывать и про эффективные методы резервного копирования. Ежедневный бэкап и хранение резервных копий на внешних альтернативных защищенных площадках особенно важны для облачных сред.**

**Кроме того, если клиент шифрует данные до размещения в облаке, стоит заранее позаботиться о безопасности хранения ключей шифрования.**

**Как только они попадают в руки злоумышленнику, с ними становятся доступны и сами данные, потеря которых может быть причиной серьезных последствий.**

1. **Недостаточная осведомленность**

**Организации, которые переходят в облако без понимания облачных возможностей, сталкиваются с рисками. Если, к примеру, команда разработчиков со стороны клиента недостаточно знакома с особенностями облачных технологий и принципами развертывания облачных приложений, возникают операционные и архитектурные проблемы.**

**CSA напоминает о необходимости понимать функционирование облачных сервисов, предоставляемых поставщиком услуг. Это поможет ответить на вопрос, какие риски берет на себя компания, заключая договор с хостинг-провайдером.**

1. **Злоупотребление облачными сервисами**

**-Облака могут использоваться легитимными и нелегитимными организациями.**

**-Цель последних — использовать облачные ресурсы для совершения злонамеренных действий:**

* **запуска DDoS-атак, отправки спама,**
* **      распространения вредоносного контента и т. д.**

**-Поставщикам услуг крайне важно уметь распознавать таких участников, для чего рекомендуется детально изучать трафик и использовать инструменты мониторинга облачных сред.**

1. **DDoS-атаки**

**Несмотря на то что DoS-атаки имеют давнюю историю, развитие облачных технологий сделало их более распространенными.**

**В результате DoS-атак может сильно замедлиться или вовсе прекратиться работа значимых для бизнеса компании сервисов.**

**Известно, что DoS-атаки расходуют большое количество вычислительных мощностей, за использование которых будет платить клиент.**

**Несмотря на то что принципы DoS-атак, на первый взгляд, просты, необходимо понимать их особенности на прикладном уровне:**

**они нацелены на уязвимости веб-серверов и баз данных.**

**Облачные поставщики, безусловно, лучше справляются с DoS-атаками, чем отдельно взятые клиенты. Главное — иметь план смягчения атаки до того, как она произойдет.**

1. **Совместные технологии, общие риски**

**-Уязвимости в используемых технологиях — достаточная угроза для облака.**

**-Поставщики облачных услуг предоставляют виртуальную инфраструктуру, облачные приложения, но если на одном из уровней возникает уязвимость, она влияет на все окружение.**

**-CSA рекомендует использовать стратегию «безопасности в глубину»:**

* **внедрять механизмы многофакторной аутентификации,**
* **системы обнаружения вторжений,**
* **придерживаться концепции сегментирования сети,**
* **принципа предоставления наименьших привилегий.**

**96. Проблемы безопасности связанные с инфраструктурой виртуальных машин. Угрозы безопасности по отношению к ВМ**

По отношению к ВМ имеются следующие угрозы их безопасности:

* Несанкционированное изменение файлов образов ВМ;

Виртуальные машины подвержены угрозам безопасности, как когда они работают, так и когда они выключены. Когда виртуальная машина выключена, она доступна в виде файла образа виртуальной машины.

Этот файл образа подвержен нескольким угрозам безопасности:

 например заражению вредоносным ПО.

Кроме того, при отсутствии соответствующих мер безопасности файлы образов виртуальных машин могут использоваться хакерами для создания новых неавторизованных виртуальных машин.

Также можно исправить файлы образов виртуальных машин, чтобы заразить виртуальные машины, создаваемые с использованием этих файлов.

Безопасность ВМ может быть поставлена под угрозу даже во время ее миграции, при этом возможны такие атаки, как:

как перехват ВМ и несанкционированная модификация.

Одна из техник, которые можно использовать для защиты файлов образов виртуальных машин, - это зашифровать их, когда они выключены или переносятся.

* Кража ВМ;

Кража виртуальной машины выполняется злоумышленником путем копирования или перемещение виртуальной машины несанкционированным образом.

В основном это возможно из-за неадекватного (не безопасного) управления файлами виртуальных машин.

Кража ВМ может оказаться фатальной, если украденная ВМ содержит конфиденциальные данные клиента.

Один из способов ограничить кражу виртуальных машин - наложить на файлы виртуальных машин ограничения на копирование и перемещение. Такие ограничения эффективно привязывают виртуальную машину к определенной физической машине таким образом, что даже если есть несанкионированная копия машины, она не будет работать на любой другой физической машине. Виртуальная машина с таким уровнем ограничений копирования и перемещения не может работать на гипервизоре, установленном на любой другой физической машине.

* Угон ВМ;

Помимо кражи ВМ, на уровне ВМ может возникнуть еще одна угроза, известная как «побег ВМ».

Обычно виртуальные машины инкапсулируются в отдельный образ и изолированы друг от друга и от базового родительского гипервизора.

В обычных сценариях для гостевой ОС и приложений, работающих на ней, нет механизма, позволяющего выйти за границы виртуальной машины и напрямую взаимодействовать с гипервизором.

Процесс выхода из виртуальной машины и взаимодействия с гипервизором называется выходом из виртуальной машины.

Поскольку гипервизор контролирует выполнение всех виртуальных машин, эта атака позволяет злоумышленнику получить контроль над любой другой виртуальной машиной.

* Атаки между ВМ;

На одной физической машине (сервере виртуализации) как правило работают несколько виртуальных машин. Таким образом, если безопасность одной виртуальной машины скомпрометирована, существует очень большая вероятность того, что безопасность других виртуальных машин, работающих на той же физической машине, будет скомпрометирована.

В одном сценариев злоумышленник может скомпрометировать одну гостевую виртуальную машину, которая затем может быть использована для атак на другие виртуальные машины, работающим на той же физической машине.

Решение:

Чтобы предотвратить возникновение таких сценариев, очень важно иметь брандмауэры и системы обнаружения вторжений в сети виртуальных машин, а также и на самих ВМ, которые могут обнаруживать и предотвращать вредоносную активность на уровне виртуальных машин.

* Мгновенный пролом.

Виртуальные машины имеют некоторые уязвимости, которых нет на физических машинах.

В основном это связано с технологиями, которые используются для  снабжения их ресурсами, использования и отключения ВМ.

Иногда эти циклы повторяются очень часто. Такая частая активация и деактивация виртуальных машин может создать проблемы для поддержания их систем безопасности в состоянии полного обновления. В результате в защите ВМ возникают «мгновенные дыры», что ведет к повышению опасности «взлома» ВМ.

Также существует вероятность того, что новые виртуальные машины могут быть клонированы и созданы из виртуальных машин, которые имеют уязвимости.

Одним из решений этой проблемы может быть выделенная виртуальная машина безопасности на каждой физической машине для автоматического обновления всех виртуальных машин, работающих на этой физической машине, всеми последними обновлениями безопасности.