## **Клиент-серверная модель вычислений. Понятие и свойства облачных технологий. Преимущества и недостатки.**

В основе облачных технологий лежат **облачные вычисления** – особый способ предоставления услуг и вычислительных ресурсов, которые в свою очередь основаны на различных способах организации распределенных вычислений. В основе организации облачных вычислений лежат два базисных **принципа** организации **распределенных** **вычислений**:

• возможность разделения решаемых задач между несколькими доступными вычислителями;

• возможность разместить вычислители, участвующие в решении задач, в местах, физически отдаленных друг от друга на значительные расстояния.

Когда речь заходит об облачных технологиях, то обычно подразумеваются следующие взаимосвязанные с ними аспекты:

• использование распределенных вычислений, которые должны быть определенным образом организованы;

• распределенные вычисления выполняются в глобальных вычислительных сетях (хотя те же самые технические и организационные решения могут применяться и в локальных вычислительных сетях, о чем еще будет сказано отдельно);

С **организационной точки** зрения облачные вычисления всегда объединяют две стороны:

• Поставщик услуг (*провайдер*) – тот, кто предоставляет услуги в процессе организации облачных вычислений. Как правило, провайдером является компания, имеющая необходимые технические ресурсы для предоставления соответствующих услуг.

• Потребитель услуг (*клиент*) – тот, кто пользуется услугами посредством облачных вычислений. В качестве клиента могут выступать отдельные физические лица или компании, имеющие потребность в данных услугах.

С **технической стороны** организации облачных вычислений можно выделить следующие участвующие в них устройства:

• ***Сервер*** – вычислительное устройство, предоставляющее услуги. Сервер может быть один или несколько – тогда они объединены в серверную часть облачных вычислений.

**• *Клиентская часть*** (клиентское устройство, клиент) – вычислительное устройство, потребляющее услуги и пользующееся ресурсами серверной части.

Согласно требованиям Национального института стандартов и технологий США зафиксированы следующие **основные характеристики** облачных вычислений:

• ***Самообслуживание по требованию*** (self service on demand, SSOD). Данное требование подразумевает, что потребитель должен иметь возможность самостоятельно определять и изменять вычислительные потребности;

• ***Универсальный доступ по сети***. Требование регламентирует тот факт, что все предоставляемые услуги должны быть доступны всем потребителям через глобальную сеть передачи данных вне зависимости от типа используемых ими клиентских устройств.

• ***Объединение ресурсов*** (resource pooling). Требование подразумевает, что провайдер (поставщик услуг) должен иметь возможность при обслуживании большого числа потребителей объединять имеющиеся ресурсы в единый пул для динамического перераспределения мощностей между потребителями в условиях постоянного изменения спроса на мощности.

• ***Эластичность***. Это требование указывает, что со стороны потребителя должна быть предусмотрена возможность предоставления, отключения, расширения, сужения услуг в любой момент времени без дополнительных издержек на взаимодействие с провайдером (в том числе в автоматическом режиме).

• ***Учет потребления***. Согласно данному требованию провайдер (поставщик услуг) должен автоматически исчислять потребленные ресурсы на определенном уровне абстракции и на основе этих данных он должен оценивать объем предоставленных потребителям услуг.

**Преимущества облачных технологий:**

• *Недорогие компьютеры для пользователей*. Пользователям нет необходимости покупать дорогие компьютеры, с большим объемом памяти и дисков, чтобы использовать программы через веб-интерфейс.

• *Увеличенная производительность пользовательских компьютеров*. Так как большая часть программ и служб запускаются удаленно в сети Интернет, пользовательские компьютеры с меньшим числом программ быстрее запускаются и работают.

• *Увеличение доступных вычислительных мощностей*. Пользователи могут запускать более сложные задачи, с большим количеством необходимой памяти, места для хранения данных, тогда, когда это необходимо.

• *Устойчивость данных к потере* или краже оборудования.

• *Мобильная доступность* информации.

**Недостатки:**

• *Постоянное соединение* с сетью Интернет.

• Не все программы или их свойства *доступны* *удаленно*. Если сравнивать программы для локального использования и их "облачные" аналоги, последние пока проигрывают в функциональности.

• *Безопасность* данных может быть под угрозой. Здесь ключевым является слово "может". Все зависит от того, кто предоставляет "облачные" услуги. Если этот кто-то надежно шифрует данные, постоянно делает их резервные копии, уже не один год работает на рынке подобных услуг и имеет хорошую репутацию, то угрозы безопасности данных может никогда не случиться.

## **Модели облачных технологий (Saas, PaaS, IaaS, DaaS, FaaS, HaaS …). Особенности и примеры.**

***«HaaS» - «Hardware as a Service»*** («аппаратные средства как услуга»)

*Особенность:*

Предоставление физических вычислительных мощностей потребителю на начальных этапах развития вычислительных технологий.

Однако при современном уровне развития информационных технологий очень быстро стало понятно, что предоставлять потребителям реальные (физические) вычислительные мощности неудобно и неэффективно даже посредством организации сетевого доступа к ним. Поэтому термин «HaaS» практически исчез с рынка информационных технологий. Ему на смену пришел следующий термин:

***«IaaS» - «Infrastructure as a Service»*** («инфраструктура как услуга»)

Предоставление облачных услуг по модели IaaS предполагает следующие варианты:

• Провайдер выделяет и распределяет имеющиеся ресурсы между клиентами (потребителями);

• Потребитель выбирает и использует ресурсы в пределах выделенного ему максимального объема;

• Ресурсы выделяются и предоставляются потребителю автоматически по мере необходимости.

*Особенности:*

• Потребителю предоставляется возможность использования выделенной облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетями и другими.

• Объем доступных ресурсов либо изначально определяется по соглашению с провайдером, либо выбирается самим потребителем из общего объема доступных со стороны провайдера ресурсов.

• Потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя ОС, а также любое системное и прикладное ПО.

• Контроль и управление основной физической и виртуальной инфраструктурой облака осуществляется провайдером. Провайдер контролирует предоставление услуг на низком уровне – на уровне ВМ, включая все программно-аппаратные средства.

***«PaaS» - «Platform as a Service»*** («платформа как услуга»)

*Особенности:*

• Клиент получает от провайдера не просто вычислительные ресурсы, но и некоторую системную платформу для организации процесса облачных вычислений. Минимальный состав – ОС.

• Вся информационно-технологическая инфраструктура платформы облачных вычислений целиком и полностью управляется провайдером. Им же управляется набор доступных потребителям платформ и управление их параметрами.

• Установленное системное ПО на уровне администрирования и системных настроек распространяется ответственность провайдера.

***«DaaS» - «Database/Data as a Service»*** («база данных/данные как услуга»)

*Особенности:*

• **DaaS = PaaS + СУБД**

• «Database as a Service» - провайдер предоставляет клиенту доступ к СУБД посредством облачных вычислений, но настройка СУБД, проектирование, создание, наполнение и управление БД – целиком и полностью ответственность потребителя.

• «Data as a Service» - провайдер может предоставить клиенту не только СУБД, но и некоторые доп. услуги по настройке СУБД.

***«SaaS» - «Software as a Service»*** («программное обеспечение как услуга»)

*Особенности:*

• Клиентам со стороны провайдера предоставляется возможность использовать специализированные прикладные программы.

• Управление ПО, его администрирование и настройка, обновление и техническая поддержка осуществляется провайдером.

• Провайдер должен иметь все необходимые права на ПО, либо иметь соглашение на право предоставлять облачные услуги с правообладателями этого ПО.

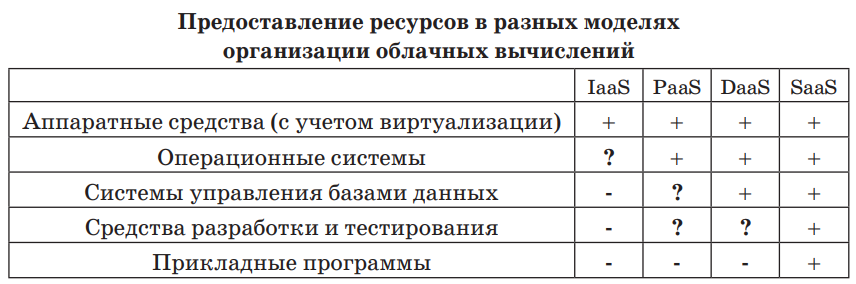
• Приложение, предоставляемое потребителям, должно быть приспособлено для использования с применением распределённых вычислений.

• Одним приложением может пользоваться несколько потребителей (в том числе и одновременно).

• Оплата за пользование приложением взимается либо в виде абонентской платы, либо путем учета объема выполненных операций.

• Техническая поддержка приложения включена в оплату.

• Обновление и модернизация программного обеспечения происходят оперативно и прозрачно для потребителей.



Другие используемые **модели организации** облачных вычислений:

• *«Storage-as-a-Service»* («хранилище как сервис») – предоставляется дисковое пространство для хранения данных.

• *«Information-as-a-Service»* («информация как сервис») – более широкий вариант трактовки DaaS.

• *«Process-as-a-Service»* («процесс как сервис» или «обработка как сервис») – предоставляется облачный ресурс, позволяющий связать воедино несколько ресурсов этого облака или нескольких облаков.

• *«Application-as-a-Service»* («приложение как сервис») – то же самое, что и «Software as a Service»

• *«Integration-as-a-Service»* («интеграция как сервис» или «взаимодействие как сервис») – предоставляется интеграционный пакет, включающий в себя интерфейсы взаимодействия между приложениями.

• *«Security-as-a-Service»* («безопасность как сервис») – услуга развертывания и настройки программных продуктов, обеспечивающих безопасное функционирование других сервисов облака.

• *«Management-as-a-Service»* («управление как сервис») – управление и настройка параметров для использования облачных сервисов.

• *«Testing-as-a-Service»* («тестирование как сервис») – предоставление средств тестирования приложений и программных продуктов на разных уровнях с использованием облачного тестового ПО.



## **3 . Глобальная инфраструктура облачного провайдера. Регионы, зоны доступности, центры обработки данных.**

Ресурсы поставщика облачных услуг физически сосредоточены в ***центрах обработки данных*** (ЦОД). ЦОД представляет собой специализированное здание, отданное под нужды провайдера и максимально снабжено для защиты аппаратуры.

При принятии решения о размещении ЦОД принимаются во внимание такие факторы, как:

• Риски окружающей среды (статистика наводнений, погодные условия, сейсмоактивность);

• Возможность сбоев;

• Текущая нагрузка на сервисы.

***Зона доступности*** – кластер ЦОД, географически расположенных рядом, связанных между собой высокоскоростными сетевыми магистралями с низкой латентностью и способным автономно функционировать с высокой степенью надежности.

Несколько зон доступности, расположенных рядом, образуют ***регион***. Зоны доступности одного региона также связаны между собой высокоскоростными сетевыми магистралями.

**4. Сервисы аренды виртуальных машин. Особенности, ценообразование**

Сервис виртуальных машин предоставляет возможность пользователям арендовать и использовать масштабируемые вычислительные мощности.

Единицей вычислительной мощности является виртуальная машина – **инстанс**.

По классификации моделей облачных вычислений данный сервис попадает в категорию «инфраструктура как услуга» (IaaS), поскольку поставщик облачных услуг управляет оборудованием, сетью, хранилищем и виртуализацией, а на откуп потребителя услуг отдается управление операционной системой виртуальной машины, системными программами, системами управления базами данных и другим программным обеспечением.

Типы инстансов EC2

Amazon EC2 предлагает большой выбор типов инстансов, которые подходят для различных нагрузок и вариантов использования

Тип определяет количество виртуальных процессоров, объем доступной памяти, наличие и емкость хранилища и другие параметры.

Определены следующие семейства типов инстансов:

1) инстансы общего назначения. Имеют сбалансированные технические характеристики. Варианты использования: сайты, репозитории кода, микросервисы, среды тестирования;

2) оптимизированные для вычислительных задач. Используют высокопроизводительные процессоры. Варианты использования: кодирование видео, нагруженные веб-сервера, научные вычисления, игровые сервера, машинное обучение;

3) оптимизированные для памяти. Используют увеличенный объем памяти. Варианты использования: базы данных в памяти, кэши, большие данные;

4) для ускоренных вычислений. Используют аппаратное ускорение вычислений, например графические процессоры. Варианты использования: 3D-рендеринг, вычисления общего назначения на графических процессорах (GPGPU), кодирование видео, распознавание речи, машинное обучение;

5) оптимизированные для хранения. Используют высокопроизводительное локальное хранилище большого объема и высокой пропускной способности. Варианты использования: распределенные вычисления, сетевые файловые системы, обработка журналов, большие данные, NoSQL базы данных, реляционные базы данных. В каждом из семейств типов инстансов есть несколько типов, различающихся своим предназначением, а внутри типа есть выбор из нескольких размеров инстансов, который можно сделать исходя из оптимизации стоимости и объемов вычислительных нагрузок.

Разумеется, тип инстанса и его размер влияют на цену аренды инстанса. Помимо этого, на цену влияют следующие факторы:

• тип и объем подключенного хранилища EBS;

• плата за лицензию операционной системы;

• плата за лицензию программного обеспечения (например, системы управления базами данных);

• регион, в котором арендуется инстанс.

Жизненный цикл инстанса

Инстанс EC2 в процессе работы может находиться в нескольких разных состояниях.

Когда инстанс запускается, то он переходит в состояние «ожидание» (pending). Плата за инстанс, находящийся в этом состоянии, не начисляется. Как только запуск инстанса завершается, он переходит в состояние «запущен» (running), после чего становится готов к использованию.

Если инстансу была дана команда остановки, он переходит в состояние «остановка», а затем в состояние «остановлен». За остановленные инстансы плата не взимается, однако она будет взиматься за тома блочного хранилища, ассоциированные с ними. Следует особо отметить, что после остановки инстанса стираются все данные хранилища инстанса (instance store). Если целью является сохранение данных при остановке инстанса, необходимо использовать тома блочного хранилища. Для инстансов, у которых нет подсоединенных к ним томов блочного хранилища, остановка невозможна.

Перезагрузка инстанса эквивалентна операции перезагрузки средствами операционной системы. В период перезагрузки инстанс находится в состоянии «перезагрузка» (rebooting), при этом сохраняются данные на хранилище инстанса, публичный IP-адрес, а также взимается плата.

Удаление инстанса (instance termination) приводит к удалению виртуальной машины и всех ассоциированных с ним ресурсов: блочного хранилища, хранилища инстанса, публичного IP-адреса.

На время выполнения удаления инстанс переходит в состояние «удаление» (shutting-down), при этом плата перестает взиматься.

Оплата EC2

Инстансы по требованию

В целях оптимизации затрат на использование вычислительных сервисов AWS предлагает несколько схем начисления оплаты за аренду виртуальных машин. Рассмотрим основные из них.

Для инстансов, арендованных по этой схеме, действует посекундная тарификация, начинающаяся с момента перехода инстанса в состояние «запущен» и заканчивающаяся при переходе в состояние «остановка» или «удаление». Данный тип тарификации имеет самую высокую цену и предназначен для непродолжительного использования вычислительных ресурсов, например, для тестирования разрабатываемых приложений или для запуска приложений с кратковременной рабочей нагрузкой.

Зарезервированные инстансы

Зарезервированные инстансы приобретаются на срок год или три года. От этого срока, а также от наличия и размера авансового платежа зависит скидка данного способа тарификации по сравнению с оплатой инстансов по требованию.

Спотовые инстансы

Спотовые инстансы позволяют использовать свободные вычислительные ресурсы поставщика облачных услуг. В этой модели пользователь устанавливает цену инстанса, которую он готов заплатить. AWS осуществляет мониторинг нагрузку на ресурсы и динамически корректирует цену инстанса. Если она опускается ниже той, которую указал пользователь, данному пользователю выделяется инстанс. Если цена повышается обратно – инстанс отбирается в течение 120 секунд. При использовании спотовых инстансов оплата за неполный час не взимается.

**5.** **Понятие масштабирования. Вертикальное и горизонтальное масштабирование.**

**Горизонтальное масштабирование приложений без состояния.**

Если система рассчитана на рост, при проектировании ее архитектуры должны быть приняты решения, поддерживающие масштабирование. Архитектура такой системы должна поддерживать увеличение количества данных или пользователей без значительного снижения производительности. Причем масштабирование с использованием приращения мощности должно приводить к пропорциональному увеличению способности выдерживать возросшую нагрузку, а приращение стоимости масштабированной инфраструктуры должно быть сопоставимо с увеличением извлекаемой прибыли.

**Вертикальное масштабирование** Вертикальное масштабирование производится путем увеличения технических характеристик отдельного ресурса, например апгрейдом сервера. В случае облачной инфраструктуры в качестве примера можно привести замену типа инстанса на более производительный (к примеру, инстанса виртуальной машины или управляемого инстанса базы данных). При этом можно увеличить характеристики оперативной памяти, центрального процессора, пропускной способности ввода-вывода и т. д. Такой тип масштабирования не всегда эффективен по стоимости, не увеличивает высокодоступность, а самое главное – имеет четкие пределы в виде максимально доступной мощности ресурса. Тем не менее в ряде случаев этот тип масштабирования приемлем, поскольку чрезвычайно прост и может быть выгоден в краткосрочной перспективе.

**Горизонтальное масштабирование.** Горизонтальное масштабирование заключается в увеличении количества ресурсов, обслуживающих нагрузку (к примеру, увеличение количества дисков в хранилище или количества серверов). Такой тип масштабирования практически не имеет предела, легко реализуем в рамках облачной инфраструктуры, но подходит не для всех приложений и архитектур.

**Состояние**. Когда пользователи или другие сервисы взаимодействуют с приложением, они могут порождать так называемое состояние в форме сессии. Сессия – данные, хранимые на сервере и отождествленные с конкретным клиентом, которые сохраняются между запросами этого клиента к приложению. Если сессия не сохраняется, то речь идет о приложении без состояния (stateless). Например, приложение, которое при одинаковых входных данных всегда выдает одинаковые выходные данные вне зависимости от того, какой пользователь с ним работает, является приложением без состояния.

**Горизонтальное масштабирование приложений без сохранения состояния.** Приложения без сохранения состояния прекрасно горизонтально масштабируются, поскольку любой из имеющихся ресурсов может обработать запрос. Данные сессий не хранятся, и вычислительные ресурсы могут добавляться при увеличении нагрузки или удаляться, когда в них более нет необходимости. Более того, ресурсы не обязаны быть осведомлены друг о друге. Единственная задача, которую нужно решить для обеспечения горизонтального масштабирования приложения без сохранения состояния, – распределение нагрузки.

**6. Горизонтальное масштабирование приложений с сохранением состояния.**

Зачастую, невозможно спроектировать приложение таким образом, чтобы оно не хранило сессии клиентов. В таком случае речь идет о приложениях с сохранением состояния (stateful). В качестве примера можно привести необходимость хранить данные о том, залогинен ли пользователь, с целью предоставления ему персонализированного контента на основании его предыдущих действий либо необходимость реализации сложных бизнес-процессов, включающих несколько шагов, где очередной шаг требует информации о действиях, выполненных в рамках предыдущих. Хорошей практикой при проектировании архитектуры сложных программных систем является обособлять их части в виде компонентов и не сохранять состояние локально в тех компонентах, где это не является очевидно необходимым.

**Масштабирование веб-приложений.** В качестве примера рассмотрим веб-приложения, клиентом для которых является веб-браузер. В данном случае механизмом сохранения состояния в приложении, то есть сессии пользователя, являются cookie. Так, веб-приложения могут хранить между запросами пользователя информацию об истории его просмотров, состоянии корзины в Интернет-магазине. Cookie хранятся на клиенте и передаются на сервер с каждым последующим запросом. Если вся перечисленная информация сохраняется в них, то, с одной стороны, это увеличивает накладные расходы на передачу данных, поскольку cookie передаются с каждым запросом, а с другой – создает угрозу безопасности, так как пользователь будет иметь возможность своевольного редактирования этих данных перед отправкой их на сервер. Большинство платформ разработки приложений предлагают механизмы, которые позволяют хранить информацию о сессии пользователя на сервере. При этом в cookie сохраняется только уникальный идентификатор сессии, что решает вышеозначенные проблемы. Тем не менее при таком подходе в случае, если данные сессии хранится на сервере локально, приложение является приложением с сохранением состояния. Очевидным выходом из сложившейся ситуации является использование базы данных, размещенной на отдельном ресурсе, для сохранения сессии. При этом сервер приложения освобождается от состояния, что облегчает его масштабирование. Еще одним примером является возможная необходимость хранения больших файлов (например, загрузок пользователя). Для обеспечения отсутствия состояния целесообразно производить сохранение таких файлов не локально на серверах приложения, а к примеру, в объектном хранилище. Тем не менее некоторые компоненты системы не могут быть избавлены от состояния ни при каких обстоятельствах. Самым очевидным примером являются базы данных. Подходы к их масштабированию описаны в соответствующем разделе.

**Привязка к сессии.** Существуют особые варианты использования облачных систем, когда пользователю необходимо в течение длительного времени быть подключенным к одному и тому же серверу. К примеру, реализация серверной части многопользовательских игр, осуществляющих взаимодействие игроков в реальном времени, предполагает, что все задействованные в игровой сессии клиенты будут достаточно долго подключены к одному игровому серверу. Это необходимо для устранения лагов и синхронизации действий игроков. Неизбежным побочным эффектом такого подхода является ситуация, когда сервер выходит из строя, и подключенные к нему игроки оказываются отключены и теряют данные, относящиеся к текущей сессии, то есть не сохраненные на сервере баз данных. Горизонтальное масштабирование таких компонентов возможно с использованием подхода, известного как привязка к сеансу (session affinity) или липкая сессия (sticky session). При этом все действия, связанные с одним сеансом, привязываются к конкретному ресурсу. В облачных системах такое реализуется специальной настройкой балансировщика нагрузки, который может маршрутизировать HTTP-трафик, перенаправляя его всегда на тот же ресурс, основываясь на идентификаторе сессии клиента, передаваемом в его cookie. Если речь идет не о HTTP, то можно реализовать липкие сессии на основании анализа IP-адреса клиента либо реализовывать логику балансировки нагрузки на стороне клиента, при этом предусмотрев на стороне сервера API для обнаружения эндпоинтов.

Эндпоинт (в переводе с англ. — конечная точка) представляет собой некий шлюз, который соединяет серверные процессы приложения с внешним интерфейсом. Простыми словами, это адрес, на который отправляются сообщения.

**7. Компьютерные сети. Адресация, подсети, маршрутизация. Преобразование адресов.**

В терминах сетевого взаимодействия любое вычислительное устройство, подключенное к сети, принято называть хостом.

Каждый хост в сети, работающей по протоколу IP, уникально идентифицируется IP-адресом. На сегодняшний день сосуществуют две версии протокола, различающиеся, помимо прочего, системой адресации.

В версии протокола IPv4 (разработана в 1981 г.) адрес имеет длину 32 бита. Адресное пространство такого типа адресации составляет примерно 4 млрд адресов.

В версии протокола IPv6 (разработана в 1996 г.) адрес имеет длину 128 бит. При этом адресное пространство составляет 340 ундециллионов адресов (один ундециллион равен ).

IANA (Internet Assigned Numbers Authority) – организация, занимающаяся управлением адресными пространствами в Интернете, доменными именами верхнего уровня, протоколами Интернета.

IP-адрес – целое 32-битное число. Для удобочитаемости оно записывается в виде 4 целых чисел, разделенных точками, каждое из которых означает один байт адреса (октет).

Сети логически разделяются (сегментируются) на подсети, что может преследовать несколько целей:

• оптимизация трафика в сети (снижается нагрузка на сетевые устройства за счет устранения ненужного в том или ином сегменте трафика);

• повышение сетевой безопасности (уменьшается периметр распространения угрозы за счет изоляции сегментов сети).

IP-адрес делится на 2 части:

• адрес сети (левая часть);

• адрес хоста в сети (правая часть).

Количество бит в правой части определяет максимальное количество устройств, которые могут быть адресованы в данной подсети.

*Например, выше адресом сети может быть 172.16.0.0, а адресом хоста в сети – 37.5. В данном случае под адрес сети отводится 16 бит и столько же – под адреса хостов в сети. Это означает, что сеть может иметь не более 65536 адресов (). При этом хостов в данной сети может быть не более 65534, поскольку нулевой адрес (172.16.0.0) резервируется под адрес подсети, а последний (172.16.255.255) – под широковещательный адрес.*

В настоящее время повсеместно применяется бесклассовая адресация (Classless Inter-Domain Routing, CIDR), при использовании которой длина частей выбирается произвольно, что позволяет гибко настраивать сегментирование и адресацию в подсетях.

Если нужно организовать подсети в рамках существующей сети, то из правой части IP-адреса выделяется необходимое количество бит, которое становится адресами подсетей данной сети.

*Для примера, если в IP-адресе 172.16.2.15 третий октет выделить под адреса подсети, то это будет означать, что сеть 172.16.0.0 будет содержать не более 256 () подсетей, каждая из которых может содержать не более 256 адресов (254 хоста). При этом для приведенного IP-адреса адресом подсети будет 172.16.2.0, а адресом хоста – 15.*

Количество бит IP-адреса, отводящихся под адрес подсети, определяется маской подсети. Маска подсети – 32-битное число, имеющее в своем двоичном представлении единицы в тех разрядах, которые относятся к адресу подсети, и нули – в разрядах, отводящихся под адреса хостов сети. Маска подсети может записываться в виде четырех октетов как IP-адрес либо как часть записи IP-адреса, в этом случае после самого адреса через косую черту указывается количество единичных бит в маске (так называемая CIDR-нотация). Для предыдущего примера, где под адрес подсети выделяется 24 бита адреса, маска подсети может быть записана следующим образом:

• 255.255.255.0

• 172.16.2.15/24

**Сетевая маршрутизация**

Сетевая маршрутизация происходит на сетевом уровне модели

OSI, опираясь на IP-адреса узлов сети. Данные в IP-сетях передаются по каналам связи в виде IP-пакетов, которые, помимо полезной нагрузки и некоторых служебных данных, содержат IP-адреса отправителя и получателя. При этом маршрутизация, то есть выбор пути передачи пакета, происходит сначала на узле-отправителе пакета, затем – на узлах-маршрутизаторах.

Узел-отправитель вычисляет, расположен ли получатель в той же подсети, что и он сам (для этого он применяет к своему адресу и адресу получателя маску подсети и сравнивает получившиеся адреса подсетей), если это так – пакет отправляется ему напрямую.

Если получатель расположен в другой подсети, пакет отправляется на шлюз подсети отправителя, адрес которого указан в настройках его сетевого интерфейса. Задача шлюза – перенаправить пакет в сеть получателя. Таким образом, шлюз выполняет функцию сетевой маршрутизации, то есть является маршрутизатором.

Основой работы сетевого маршрутизатора являются таблицы маршрутизации, содержащие сопоставления адресов сетей с адресами маршрутизаторов, на которые необходимо отправить пакет для доставки в соответствующую сеть. Также таблица маршрутизации обязана содержать маршрут по умолчанию – инструкции о том, куда отправлять пакеты, адресов сетей получателей которых нет в таблице.

**Преобразование сетевых адресов**

Преобразование сетевых адресов (Network Address Translation, NAT) – механизм, использующийся в IP-сетях для замены адресов при передаче пакетов между сетями, имеющими разные адресации.

Чаще всего встречается при необходимости подключения локальной сети к Интернету, но может использоваться и внутри сегментов локальных сетей.

Если шлюз осуществляет трансляцию адресов, то он заменяет локальные адреса хостов-отправителей при пересылке пакетов на адрес своего внешнего сетевого интерфейса. При обратной пересылке пакета адрес сетевого интерфейса шлюза в нем заменяется на локальный адрес получателя, то есть происходит обратная замена.

Важным следствием описанного механизма является тот факт, что хосты внешней (глобальной) сети не могут осуществлять подключение к хостам внутренней (локальной) сети.

**8. Сервис VPC (Virtual Private Cloud). Возможности для настройки сетевой**

**инфраструктуры. Шлюзы, таблицы маршрутизации**

Любые облачные ресурсы (например, инстансы виртуальных машин) размещаются и взаимодействуют в виртуальной сети.

Рассмотрим особенности создания и настройки виртуальной сети средствами сервиса Amazon VPC.

При создании сети указывается:

• имя сети (тег Name);

• регион, в котором создается сеть (при этом сеть распространяется на все зоны доступности данного региона);

• блок адресов.

После создания виртуальной сети необходимо создать подсети для этой сети. В отличие от традиционных типов сетей, в облачной инфраструктуре подсети преследуют иные цели:

• обеспечение высокой доступности

Есть возможность создать вторую подсеть в другой зоне доступности и разместить в ней дублирующие ресурсы. Если первая зона доступности окажется недоступна, можно будет задействовать ресурсы в другой зоне доступности (второй подсети);

• предоставление разных свойств сетевого доступа к ресурсам

Для каждой сети можно задать настройки безопасности, таким образом контролируя доступ к расположенным в них ресурсам.

При создании подсети указывается:

1) виртуальная сеть, для которой создается подсеть;

2) зона доступности, в которой создается подсеть (подсеть должна принадлежать одной конкретной зоне доступности);

3) блок адресов, назначаемый сети (должен входить в блок адресов виртуальной сети).

Адресация подсетей виртуальной сети использует те же принципы, что и адресация в традиционных сетях - (Вопрос 7)

**Маршрутизация в виртуальной сети**

Для виртуальной сети и каждой ее подсети необходимо определить таблицы маршрутизации.

Таблица маршрутизации состоит из маршрутов (строки таблицы), каждый маршрут включает две части:

• назначение (destination, диапазон IP-адресов, куда направляется трафик);

• шлюз (target, IP-адрес шлюза).

По умолчанию для виртуальной сети и каждой ее подсети создается таблица, содержащая маршрут для локального трафика. Например, для подсети 10.0.0.0/16 таблица маршрутизации с маршрутом по умолчанию будет иметь вид:

|  |  |
| --- | --- |
| Назначение | Шлюз |
| 10.0.0.0/16 | local |

По умолчанию подсети, созданные в сервисе Amazon VPC, не имеют связи с Интернетом (как для входящего, так и для исходящего трафика). В случае если хотя бы одна подсеть должна быть связана с Интернетом, необходимо:

• создать Интернет-шлюз

Интернет-шлюз (Internet Gateway, IGW) – бесплатный масштабируемый компонент VPC, который позволяет подключить виртуальную сеть к Интернету. Для виртуальной сети создается одиночный Интернет-шлюз. Он осуществляет маршрутизацию трафика в Интернет, преобразование сетевых адресов (NAT) для ресурсов c публичным IP-адресом;

• создать маршруты в таблицах маршрутизации для каждой имеющей связь с Интернетом подсети

Например, в приведенную выше таблицу маршрутизации можно добавить такой маршрут:

|  |  |
| --- | --- |
| Назначение | Шлюз |
| 10.0.0.0/16 | Local |
| 0.0.0.0/0 | igw-id |

В этой таблице igw-id– идентификатор Интернет-шлюза, а 0.0.0.0/0– все адреса, не подходящие под другие маршруты данной таблицы.

В терминах VPC подсеть, имеющая маршрут к Интернет-шлюзу, называется публичной. Ресурсы такой сети доступны из Интернета, а также сами имеют доступ к Интернету. Подсеть, не имеющая маршрута к Интернет-шлюзу, называется приватной.

Зачастую необходимо обеспечить доступ к Интернету ресурсов из приватной подсети. Например, это может быть полезно, чтобы дать возможность программному обеспечению скачивать обновления.

Для этой цели необходимо:

• создать NAT-шлюз

NAT-шлюз (NAT Gateway) – масштабируемый компонент VPC, который позволяет подключить приватную подсеть к Интернету.

NAT-шлюз создается в конкретной зоне доступности. Он осуществляет маршрутизацию трафика в Интернет, преобразование сетевых адресов для ресурсов c частным IP-адресом. При этом приватная подсеть не становится публичной, поскольку по-прежнему отсутствует доступ извне к ресурсам подсети.

• Создать маршруты в таблицах маршрутизации для каждой имеющей связь с Интернетом частной подсети.

Например, в приведенную выше таблицу маршрутизации для частной подсети можно добавить такой маршрут:

|  |  |
| --- | --- |
| Назначение | Шлюз |
| 10.0.0.0/16 | Local |
| 0.0.0.0/0 | nat-id |

В этой таблице nat-id – идентификатор NAT-шлюза, а 0.0.0.0/0 – все адреса, не подходящие под другие маршруты данной таблицы.

**9. Сетевая безопасность в VPC. Списки контроля доступа, группы безопасности.**

**Публичные и частные подсети.**

Важным аспектом проектирования сетей и традиционных, и виртуальных является сетевая безопасность. Прежде всего, обеспечение безопасности в этом случае заключается в настройке правил, регулирующих разрешение или запрещение того или иного трафика. В традиционных сетях программы или устройства, осуществляющие такой контроль, называются файрволлами, брандмауэрами или сетевыми экранами.

Хорошей практикой является использование многоуровневой сетевой защиты, когда тщательно настраиваются правила на уровне:

1) подсети– средствами сетевых списков контроля доступа (Network Access Control Lists, ACL);

2) сетевого ресурса– средствами групп безопасности (Security Groups);

3) хоста – средствами операционной системы хоста или установленного программного обеспечения.

**Сетевые списки контроля доступа**

Для каждой подсети виртуальной сети могут быть настроены наборы правил сетевого доступа. Данные наборы правил, по сути, являются аналогом наличия в составе данной сети сетевого экрана, осуществляющего контроль трафика.

Правила в зависимости от типа трафика, который они контролируют, делятся на входящие и исходящие.

Для каждого входящего правила указываются:

• номер правила;

• тип (определяет протокол и диапазон портов);

• протокол, к которому применяется правило (TCP, UDP, ICMP…);

• диапазон портов, на который распространяется правило;

• источник (диапазон адресов хостов-источников входящего трафика);

• разрешение или запрет соответствующего входящего трафика.

Для каждого исходящего правила указываются:

• номер правила;

• тип (определяет протокол и диапазон портов);

• протокол, к которому применяется правило (TCP, UDP, ICMP…);

• диапазон портов, на который распространяется правило;

• назначение (диапазон адресов хостов-назначений исходящего

трафика);

• разрешение или запрет соответствующего исходящего трафика.

В правилах порт – 16-битное целое неотрицательное число, на транспортном уровне OSI определяющее тип соединения меду хостами. IANA зарезервировано большое количество портов под различные протоколы и приложения, осуществляющие сетевое взаимодействие. *Например:*

*• HTTP – 80*

*• HTTPS – 443*

*• FTP – 21*

*• SSH – 22*

*• MySQL – 3306*

*• PostgreSQL – 5432*

*Рассмотрим примеры списков контроля доступа.*

*Входящие правила:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Номер* | *Тип* | *Протокол* | *Диапазон*  *портов* | *Источник* | *Разрешить/*  *запретить* |
| *100* | *IPv4 traffic* | *All* | *All* | *0.0.0.0/0* | *ALLOW* |
| *\** | *IPv4 traffic* | *All* | *All* | *0.0.0.0/0* | *DENY* |

*Исходящие правила:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Номер* | *Тип* | *Протокол* | *Диапазон*  *портов* | *Назначение* | *Разрешить/*  *запретить* |
| *100* | *IPv4 traffic* | *All* | *All* | *0.0.0.0/0* | *ALLOW* |
| *\** | *IPv4 traffic* | *All* | *All* | *0.0.0.0/0* | *DENY* |

Правила применяются сверху вниз. Если найдено подходящее правило, более нижние игнорируются.

*В данном примере разрешен весь трафик как входящий, так и исходящий (правила 100 в каждом списке). Это не является хорошей практикой настройки сетевой безопасности, поскольку создает угрозу неправомерного сетевого доступа.*

*Рассмотрим модифицированные списки контроля доступа.*

*Входящие правила:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Номер* | *Источник* | *Протокол* | *Диапазон*  *портов* | *Разрешить/*  *запретить* |
| *100* | *0.0.0.0/0* | *TCP* | *443* | *ALLOW* |
| *110* | *192.0.2.0/24* | *TCP* | *3389* | *ALLOW* |
| *\** | *0.0.0.0/0* | *All* | *All* | *DENY* |

*Исходящие правила:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Номер* | *Источник* | *Протокол* | *Диапазон*  *портов* | *Разрешить/*  *запретить* |
| *100* | *0.0.0.0/0* | *TCP* | *1025–65535* | *ALLOW* |
| *\** | *0.0.0.0/0* | *All* | *All* | *DENY* |

*В данном примере разрешается входящий трафик по протоколу HTTPS с любых адресов, а также входящий RDP-трафик (порт 3389) с некоего доверенного диапазона адресов. Остальной входящий трафик запрещается.*

*Также разрешается исходящий трафик от серверов к клиентам по ряду портов. Остальной исходящий трафик запрещается.*

**Группы безопасности**

Группы безопасности (security groups), так же, как и сетевые списки контроля доступа, содержат правила контроля трафика. Но правила в группах безопасности применяются не к подсетям, а к конкретным облачным ресурсам или группам ресурсов. Также одному ресурсу может быть назначено несколько групп безопасности.

По умолчанию создаваемые группы безопасности настроены таким образом, что они разрешают весь исходящий трафик и блокируют весь входящий.

Хорошей практикой является тщательный анализ прав доступа одних облачных ресурсов виртуальной сети к другим, а также создание и соответствующая настройка необходимого количества групп безопасности для этих ресурсов.

*Например, если приложение использует многоуровневую архитектуру, компоненты каждого слоя должны иметь только такой сетевой доступ, который необходим для их функционирования. К примеру, группа безопасности слоя представления (веб-интерфейс) может иметь правило, разрешающее входящий трафик на порт 443 с любых IP-адресов. Так можно обеспечить возможность входящего подключения к серверу по протоколу HTTPS. Группа безопасности слоя приложения может иметь правило, разрешающее входящий трафик на порт 80 для источника группы безопасности слоя представления. Наконец, группа безопасности слоя базы данных может иметь правило, разрешающее входящий трафик на порт 3306 (порт MySQL), в качестве источника трафика при этом указывая группу безопасности слоя приложения.*

**10. Служба DNS. Назначение, принципы работы. DNS-маршрутизация.**

В основе современной адресации лежит протокол IP, определяющий адрес, как последовательность из 4 байтов или 16 байтов для версии протокола 4 (IPv4) и для версии протокола 6 (IPv6) соответственно.

Например, адрес IPv4– 173.194.73.18, а IPv6– 2a00:1450:4010:c0d::12. Оба указанных адреса соответствуют одному из популярнейших имен узлов – mail.google.com.

Имя узла – символьное обозначение устройства компьютерной сети, соответствующее некоторому IP-адресу. В ряде случаев удобнее использовать имена узлов при взаимодействии с различными устройствами компьютерной сети. Например, пользователю может быть сложно запомнить IP-адрес, особенно IPv6, или IP-адрес определяется и изменяется сетевыми настройками (DHCP) и не является постоянным. В таком случае необходимо осуществлять дополнительную передачу информации об изменении параметров доступа к устройству всем заинтересованным сторонам некоторым способом, определенным заранее, а ряд задач не могут быть автоматизированы.

**Система работы файла hosts**

В базовом случае имя узла или несколько имен соответствую одному IP-адресу. Изначально данное соответствие определялось записью в конфигурационном файле hosts.txt или hosts операционной системы, в зависимости от используемой операционной системы (MS Windows, Unix, Gnu/Linux, MacOS и т. д.), где запись имеет вид:

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

Файл hosts является текстовым файлом, а такой способ взаимодействия с именами устройств имеет ряд недостатков:

• с увеличением количества записей увеличивается сложность поддержки данных в актуальном состоянии;

• передача данных о записях осуществляется копированием файла между устройствами;

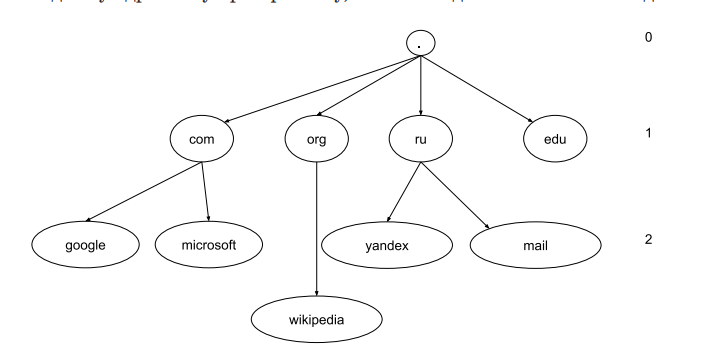
отсутствие централизованного управления записями.

**Система доменных имен**

Одним из решений существующих проблем стала разработка архитектуры системы доменных имен (Domain Name System, DNS), определенной в виде RFC 8822и RFC 8833 (Request for Comments), где задается иерархическое пространство имен с типизированными данными об узлах. У каждого узла иерархического пространства имен есть идентификатор – доменное имя. Домен – некоторая область пространства имен, соответствующая определенному уровню в иерархии.

Корневым элементом такой иерархической структуры является корневой домен «.». Домены верхнего (первого) уровня: ru, com, org, edu и т. д. Такие домены предоставляются странам или обозначают тип организации. Следующий уровень называется вторым уровнем, где расположены домены вида google.com, kernel.org и т. д. Домены второго уровня регистрируются на год в рамках одной зоны верхнего уровня.

Пример такой иерархической структуры изображен на рисунке.



Доменное имя – символьный идентификатор, определяющий область административной автономии и контроля в компьютерных сетях различного ранга в Интернете. Совокупность имен, относящаяся к одному адресному пространству, называют доменной зоной. Разделяют два типа зон: прямые и обратные. Прямые зоны задают соответствие имя-адрес, а обратные адрес-имя.

**Записи DNS**

DNS – распределенная база данных, позволяющая получать информации о различных доменных именах. DNS позволяет определить IP-адрес для некоторого доменного имени или доменное имя для некоторого IP-адреса.

При работе с DNS клиент обращается к DNS серверу с запросом определения IP адреса для доменного имени, например, example.com.

DNS сервер обращается к корневому серверу, который возвращает адрес DNS сервера, обслуживающего доменную зону com. Клиент отправляет запрос к DNS серверу доменной зоны com, который в свою очередь возвращает адрес сервера, контролирующего доменную зону example.com, который уже возвращает IP-адрес доменного имени examle.com. Следует отметить, что одному IP-адресу могут соответствовать несколько имен, а нескольким IP-адресам одно имя.

На данный момент существует 13 корневых серверов, расположенных в различных локациях.

Запись DNS состоит из следующих полей:

• NAME – доменное имя, к которому привязана или которому

«принадлежит» данная ресурсная запись,

• TYPE – тип, определяющий формат и назначение записи,

• СLASS,

• TTL (Time To Live) – допустимое время хранения записи в кэше

неответственного DNS-сервера,

• RDLEN – длина поля данных,

• RDATA – поле данных.

Существуют более 45 различных типов записей. Далее перечислены наиболее часто используемые:

• A – запись адреса, задает соответствие имени узла и IP-адреса версии IPv4.

• AAAA – запись адреса, задает соответствие имени узла и IPадреса версии IPv6.

• CNAME – каноническая запись имени узла, используется для

перенаправления на другое имя.

• MX – указывает на почтовый сервер для данного домена.

• NS – указывает на DNS-сервер для данного домена.

• PTR – обратная DNS-запись, которая связывает IP-адрес узла

с его каноническим именем.

Обратите внимание, что в конце имен при запросах стоит «.». Это не ошибка, а полное

доменной имя согласно иерархии имен, с корневым доменом.

Реализации

Сервис системы доменных имен можно реализовать несколькими основными реализациями:

• BIND (Berkeley Internet Name Domain),

• Dnsmasq1,

• Microsoft DNS Server2.

Как правило, DNS-сервер использует порт UDP/TCP 53. Программа nslookup, отправляет запросы к DNS-серверу через порт UDP 53.

Различные облачные провайдеры предоставляют услуги сервиса системы доменных имен, например:

• Azure DNS3,

• Google Cloud DNS4,

• Amazon Route 535.

**11. Облачные хранилища данных. Виды и свойства**

**Облачное хранилище данных** – модель предоставления услуг хранения данных поставщиком облачных услуг (cloud provider), где данные объединены в логические группы, которые распределены на множестве устройств хранения данных, объединенных сетью и другими компонентами, обеспечивающими доступ к данным.

Облачное хранилище включает в себя тысячи устройств хранения данных, кластеризованных по сети, распределенных файловых систем и других промежуточных хранилищ.

**Способы доступа к данным через сервисы поставщика:**

• (Compute),   
• API,  
• API CLI (Cloud Computing Infrastructure Service).

**Основные требования при использовании облачных технологий для хранения данных:**

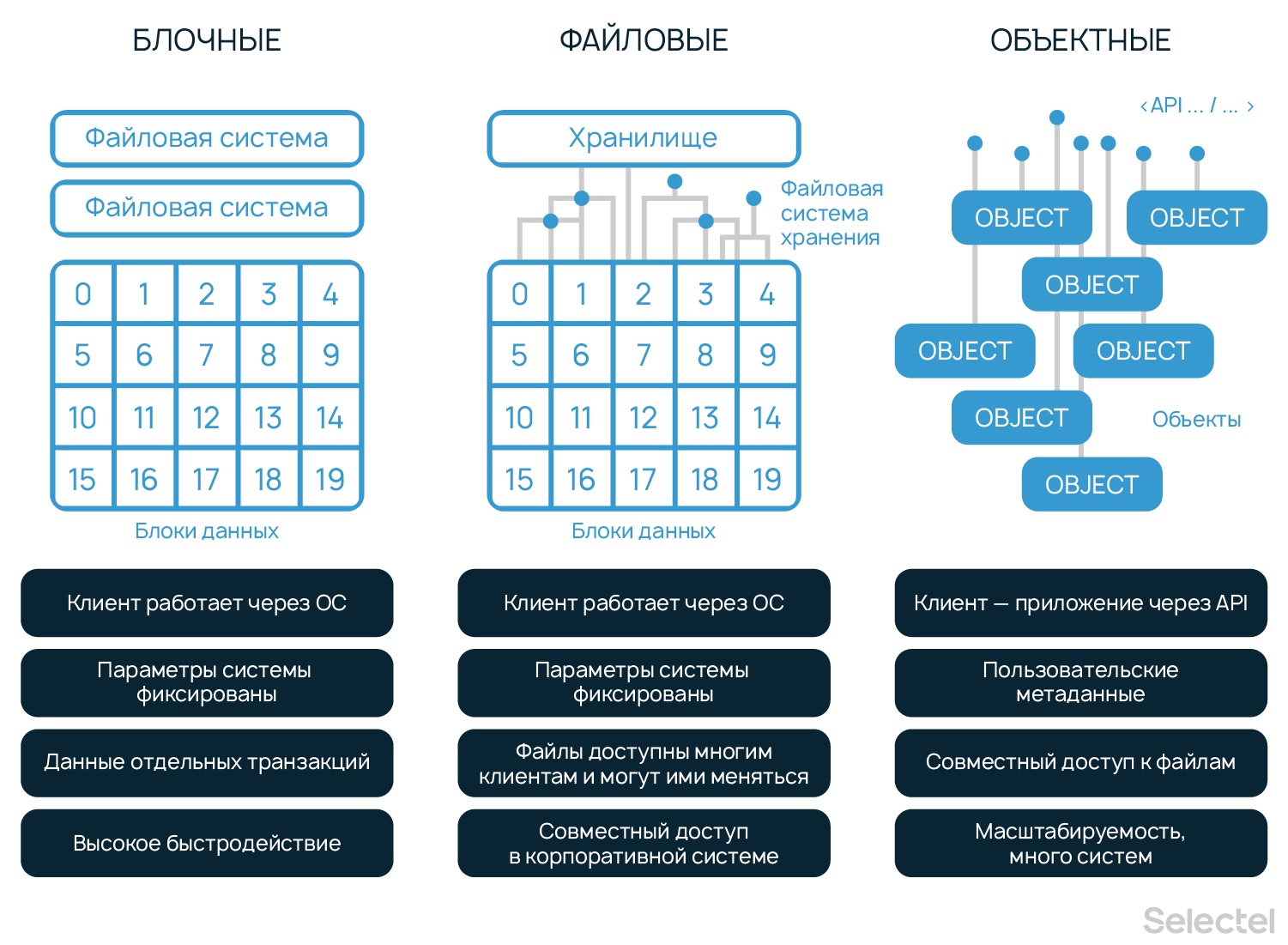
• надежность хранения данных;   
• доступность данных;   
• безопасность хранения данных.

**Виды облачных хранилищ:**

**• Блочные хранилища** - разбивают данных на части (chunks), произвольно организованные при хранении в томах (volumes) одинакового размера.

**• Файловое хранилище** - хранит и предоставляет доступ к данным в виде иерархии файлов и каталогов.

**• Объектное хранилище** - управляет данными, связывая их с метаданными.



*У каждого из перечисленного вида хранилища данных имеются свои ограничения по использованию и уникальный набор возможностей, определяющий выбор вида хранилища данных для решения конкретной задачи*.

**Блочные хранилища**

*Блочное хранилище работает аналогично хранилищу с прямым подключением (DAS) или сети хранения данных (SAN).*

Применение блочных хранилищ данных решает задачи уменьшения задержек при работе с данными, где данные разбиваются на блоки, а каждому блоку присваивается уникальный идентификатор, который позволяет системе хранения размещать более мелкие фрагменты данных там, где это наиболее удобно. Поставщики облачных услуг выделяют блочное хранилище для каждого виртуального сервера и обеспечивают относительно малые задержки при рабочих нагрузках, требующих высокой производительности.

**Файловое хранилище**

При использовании файловых хранилищ данных данные хранятся в виде многоуровневой системы файлов (файловая система) и каталогов, организованных иерархически, где отдельные файлы хранятся в каталогах, начиная с изначального корневого каталога. Для доступа к данным требуется указать полный путь к файлу или путь относительный к текущей позиции в структуре каталогов. Файловая система – одна самых старых и широко распространенных систем хранения данных, используемая для прямого доступа к файлам, а также в сетевых устройствах хранения данных NAS (Network Attached Storage).

**Этот вид облачного хранилища данных подходит в случае использования в качестве:**

• системы совместного использования файлов;   
• крупных репозиторий контента (artifactory);   
• мультимедийных хранилищ;   
• личных каталогов пользователей.

**Объектное хранилище**

Объектные хранилища данных позволяют хранить файлы как объекты, доступ к которым осуществляется с использованием HTTP API. Объекты группируются с помощью пространств имен (namespaces) в «ведра» (buckets). Последнее может содержать множество объектов, но один объект принадлежит только одному «ведру». У каждого объекта есть свой уникальный идентификатор.

Объектные хранилища позволяют хранить неструктурированные данные различных типов. У каждого объекта есть набор метаданных, характеризующих объект.

**Метаданные** – важный элемент объектных хранилищ данных, который содержит расширенную информацию об объекте *(например, время с момента создания, права доступа, данные о геолокации при создании оригинальных данных).*   
Для получения доступа к данным используются метаданные и идентификаторы.

Объектные хранилища обладают хорошей масштабируемостью, что привело к широкому применению в облачных технологиях при хранении статичных данных.   
Однако объекты не могут быть модифицированы, поэтому для изменения данных объект должен быть перезаписан полностью. Процесс перезаписи медленный, поэтому объектные хранилища малоэффективны для хранения данных традиционных баз данных.

**Основные преимущества** применения облачных хранилищ данных над традиционными локальными (on-premises) хранилищами данных:

• доступность данных;   
• отсутствие необходимости физического носителя;   
• скорость масштабирования;   
• централизованное управление данными и доступом к ним.

**Вопросы, требующие пристального внимания**, при использовании облачных хранилищ данных в сравнении с традиционными локальными (on-premises) хранилищами данных:

• безопасность данных;   
• целостность данных;   
• время и стоимость репликации данных;   
• цена миграции.

**12. Облачное объектное хранилище. Классы хранения**

**• Объектное хранилище** - управляет данными, связывая их с метаданными.

Объектные хранилища данных позволяют хранить файлы как объекты, доступ к которым осуществляется с использованием HTTP API. Объекты группируются с помощью пространств имен (namespaces) в «ведра» (buckets). Последнее может содержать множество объектов, но один объект принадлежит только одному «ведру». У каждого объекта есть свой уникальный идентификатор.

Объектные хранилища позволяют хранить неструктурированные данные различных типов. У каждого объекта есть набор метаданных, характеризующих объект.

**Метаданные** – важный элемент объектных хранилищ данных, который содержит расширенную информацию об объекте *(например, время с момента создания, права доступа, данные о геолокации при создании оригинальных данных).*   
Для получения доступа к данным используются метаданные и идентификаторы.

Объектные хранилища обладают хорошей масштабируемостью, что привело к широкому применению в облачных технологиях при хранении статичных данных.   
Однако объекты не могут быть модифицированы, поэтому для изменения данных объект должен быть перезаписан полностью. Процесс перезаписи медленный, поэтому объектные хранилища малоэффективны для хранения данных традиционных баз данных.

**Основные преимущества** применения облачных хранилищ данных над традиционными локальными (on-premises) хранилищами данных:

• доступность данных;   
• отсутствие необходимости физического носителя;   
• скорость масштабирования;   
• централизованное управление данными и доступом к ним.

**Вопросы, требующие пристального внимания**, при использовании облачных хранилищ данных в сравнении с традиционными локальными (on-premises) хранилищами данных:

• безопасность данных;   
• целостность данных;   
• время и стоимость репликации данных;   
• цена миграции.

Объектные хранилища получили широкое распространение в облачных технологиях с запуском облачного сервиса Amazon Simple Storage Service (Amazon S3) компанией Amazon в марте 2006 года. Широкое распространение и популярность сервиса привели к появлению конкурирующих сервисов объектного хранения данных у других поставщиков облачных услуг на основе S3 API. S3 API постепенно становится «де-факто» стандартом доступа к облачным объектным хранилищам данных.

**Примеры облачных объектных хранилищ данных:**

• Amazon S3,   
• Azure Blob Storage,   
• Google Cloud Storage.

**Классы хранения:**

В основном облачные объектные хранилища данных разделяют на 3 типа:

**• Стандартные** - для хранения часто используемых данных. Обеспечивая низкую задержку и высокую пропускную способность.

**• Холодные** - для хранения данных, доступ к которым осуществляется относительно редко, но при этом должен обеспечиваться быстро.

**• Ледяные** - предназначены специально для архивации данных на долгий период.

В зависимости от выбранного класса, будет изменяться стоимость хранения.  
Некоторые поставщики услуг делят классы на подклассы, тем самым пользователь может выбрать тариф, который ему больше всего подходит. Например, у Amazon в классе «холодного» хранения имеется 3 подкласса.

**13. Понятие виртуализации и контейнеризации.   
Их роль в организации программных систем. Гипервизор**

Различные технологии виртуализации получили широкое распространение в облачных технологиях, обеспечивая более гибкое и плотное использование физических вычислительных устройств. Одним из основных базовых компонентов технологии виртуализации является гипервизор, а виртуализацию называют гипервизовой виртуализацией.

**Гипервизор** – программное обеспечение, осуществляющее работу виртуальных машин с использованием той же архитектуры набора инструкций, что и оборудование, на котором гипервизор выполняется.

**Гипервизор**, или диспетчер виртуальных машин, — это процесс, обеспечивающий создание и выполнение виртуальных машин (ВМ).

**Гостевые операционные системы** - ОС, установленные в виртуальных машинах.

**Гипервизор обеспечивают:**

• изоляцию,   
• разделение ресурсов,   
• защиту,   
• безопасность гостевых операционных систем.

**Гипервизоры бывают трех типов:**

• автономные;   
• на базе основной (хостовой) операционной системы;   
• гибридные.

**Контейнеризация** же, в отличие от гипервизовой виртуализации, обеспечивает:

• изоляцию,   
• разделение ресурсов,   
• защиту,   
• безопасность,   
**на уровне пространства пользователя хостовой операционной системы.**

Контейнерную виртуализацию называют виртуализацией уровня операционной системы. Контейнерная виртуализация рассматривается как менее безопасная в сравнении с гипервизором. Однако у технологии контейнеризации есть ряд преимуществ, обеспечивающий широкое распространение технологии, особенно в облачных окружениях.

**Основное преимущество контейнеризации – меньшие накладные расходы в сравнении с традиционными технологиями виртуализации.** В отличие от гипервизовой виртуализации, контейнеризация не требует запуска отдельного уровня эмуляции или гипервизора. Вместо уровня эмуляции и гипервизора используются системные вызовы операционной системы. А сборка, передача контейнеров по сети и запуск контейнеров осуществляется быстрее, чем при использовании гипервизовой виртуализации.   
Данные преимущества прямым образом влияют на скорость масштабирование приложений, что немаловажно при работе с облачными технологиями.   
Для запуска контейнера необходимо выполнить ряд системных вызовов, отвечающих за запуск процесса, монтирование файлов образа и изоляцию выполнения процесса.

**14. Бессерверные вычисления.   
Понятие, принципы, преимущества, недостатки**

**Понятие бессерверных вычислений**

**Бессерверные вычисления *(Serverless computing)*** – метод предоставления и организации облачных услуг по принципу фактического использования сервиса в зависимости от пользовательской нагрузки, при котором облако автоматически управляет выделением вычислительных ресурсов.

В **основе** бессерверных вычислений лежит реализация шаблона **«функция как услуга»**, при котором для выполнения каждого запроса *(вызова функции)* создается отдельный контейнер или виртуальная машина, уничтожающиеся после выполнения.

В результате вычислительные ресурсы масштабируются автоматически по мере увеличения либо уменьшения нагрузки.

Фактически, бессерверные вычисления представляют собой упрощение модели облачной среды, позволяющей создавать программный продукт, не беспокоясь о серверах.

**Преимущества бессерверных вычислений**

**К преимуществам бессерверных вычислений относятся:**

• **Снижение затрат.**По причине фактической оплаты по принципу pay as you go (*pay per use*) – пользователь платит только за то, что использует.   
Работа вычислительной среды по требованию означает запуск бессерверной функции только в момент регистрации какого-то события-триггера, а по завершении операций – сворачивании всей среды до запуска следующей задачи.

• **Маштабируемость**.   
Достигается благодаря архитектуре бессерверных вычислений.

• **Скорость разработки.**   
Определяет время выхода продукта на рынок. Благодаря бессерверным вычислениям эта скорость увеличивается в разы.

**Недостатки бессерверных технологий**

**Несмотря на преимущества технологии, у нее есть и свои недостатки:**

**• Скорость обработки.**Холодный старт приложения, при котором его запуск происходит с нуля из-за подключения вычислительной нагрузки, приводит к увеличению времени на обработку запроса.

**• Короткие процессы.**Не подходит для работы долгосрочных процессов, так как потребление ресурсов будет таким же, как у традиционных продуктов, а цена – выше.

**• Ограниченное количество поддерживаемых языков программирования.**При подборе бессерверного варианта развертывания разработчик должен учитывать язык разработки модулей, входящих в его продукт с учетом поддерживаемых языков целевой платформы. Еще наиболее усложняет обстановку тот факт, что разные бессерверные платформы поддерживают разные комплекты языков, что усложняет миграцию между ними.

**• Несовместимость платформ.**Бессерверные платформы имеют разные методы формирования, развертывания, обновления и мониторинга бессерверных функций. Это усложняет миграцию проектов.

**• Расширение архитектуры, а не самостоятельное решение.**В отличие от контейнеров или виртуальных машин, которые применяются для развертывания всего приложения на целевой платформе, бессерверное решение – это метод расширения продукта. Таким образом, бессерверная реализация является частью архитектуры, а не самодостаточной архитектурой.

**• Сложная отладка.**Возможны сложности в отладке из-за модульной раздробленности приложения.

**• Болезни роста.**Возможные проблемы безопасности или недостаточной оптимизации технологии в силу относительной новизны технологии; скудные средства разработки.

**Исходя из описанных преимуществ и недостатков**бессерверные вычисления хорошо применимы в случаях таких бизнес-процессов, при которых не требуется мгновенный отклик, присутствует скачкообразный рост и спад нагрузки.

***Пример:***

*задачи, выполняемые по расписанию (создание бэкапов),*

*асинхронная отправка уведомлений (рассылок),*

*машинное обучение (дообучение на новых наборах данных),*

*ИИ-системы и подобные задачи, связанные с реагированием на события.*

*Использование бессерверных вычислений позволяет быстро выпускать продукт (или фичу) для массового потребителя, так как провайдер полностью берет на себя операционное обеспечение IТ-инфраструктуры, включая подготовку и масштабирование вычислительных кластеров, обеспечение безопасности, установку обновлений, устранение сбоев оборудования.*

**15. Реляционные облачные базы данных.   
Масштабирование, высокая доступность**

Реляционные базы данных широко применяются в составе традиционных инфраструктур, но есть ряд вариантов использования облачных систем, в которых реляционные базы данных обладают неоспоримыми преимуществами, прежде всего, в части ACID-транзакционности:

• системы планирования ресурсов предприятия (ERP-системы, Enterprise Resource Planning);   
• системы управления отношениями с клиентами (CRM-системы, Client Relationship Management);   
• коммерческие приложения;   
• финансовые приложения.

Существуют два подхода, которыми можно воспользоваться при развертывании системы управления базами данных в составе облачной инфраструктуры.

**1. Использование управляемого сервиса реляционных баз данных.**

При использовании данного подхода пользователь пользуется сервисом провайдера, предоставляющим набор услуг по управлению экземпляром системы управления базами данных, а выделение инфраструктуры, установка, настройка программного обеспечения и обслуживание экземпляра входят в обязанности провайдера. Пользователь вправе выбрать мощность инстанса *(по аналогии с мощностью инстанса виртуальной машины при использовании сервиса виртуальных машин),* реляционный движок (MySQL, MariaDB, PostgreSQL, Oracle Database, Microsoft SQL Server) и некоторые параметры конфигурации.

**Преимущества данного подхода:**

• отсутствие затрат на установку и обслуживание системы управления базами данных;   
• уверенность в качестве настройки экземпляра системы управления базами данных.

**2. Использование собственноручно настроенной системы управления базами данных, запущенной на виртуальной машине.**

В этом случае все обязанности по установке, настройке и обслуживанию системы управления базами данных ложатся на пользователя.

**Преимущества данного подхода:**

• более низкая стоимость инстанса виртуальной машины в сравнении с инстансом сервиса баз данных;   
• большая гибкость в настройке системы управления базами данных, поскольку любые ее параметры доступны для изменения, начиная с момента установки.

**Масштабирование**

Реляционные базы данных поддерживают **вертикальное масштабирование** путем перенесения на более мощный управляемый инстанс, использования более быстродействующего хранилища или хранилища большего объема. Чаще всего подобное масштабирование возможно выполнить без простоя, в процессе работы.

Помимо вертикального масштабирования, довольно легко организовать **горизонтальное масштабирование** реляционной системы управления базами данных путем создания реплик чтения. Это имеет смысл для приложений с большим количеством операций чтения, когда обслуживание их одним инстансом базы данных не удовлетворяет требованиям эффективности.

**Реплики чтения**– отдельные инстансы управляемой базы данных, представляющие собой копии исходного инстанса и реплицируемые асинхронно. Асинхронность репликации означает, что реплики чтения могут «отставать» от основного инстанса, то есть не иметь результатов нескольких последних транзакций (*так называемая задержка репликации).* Также реплики чтения не могут принимать запросов на запись, следовательно, **невозможно** с их помощью **горизонтально** масштабировать операции записи в базу данных.

Горизонтальное масштабирование записи в базу данных при исчерпании ресурсов одиночного инстанса можно выполнить, используя шардирование данных. При этом данные распределяются между несколькими инстансами системы управления базами данных. **Преимуществом** **данного подхода является возможность масштабирования записи в базу данных.**

**Высокая доступность**

Хорошей практикой при развертывании реляционной базы данных в составе облачной системы является создание синхронно реплицируемого резервного экземпляра в другой зоне доступности.

Большинство облачных провайдеров могут делать это автоматически при выборе соответствующей опции. Резервная реплика не является активной базой данных и не получает запросов от приложений. В случае отказа основного экземпляра или сбоя зоны доступности (*в общем случае – проблемы подключения к основному экземпляру*) автоматически задействуется резервный экземпляр базы данных, повышенный до основного. Он будет иметь тот же адрес, что и вышедший из строя экземпляр, и начнет принимать запросы, что позволит приложениям продолжить работу без необходимости ручного вмешательства.

Чтобы высокая доступность обеспечивалась и после возникновения неисправности, создается новая синхронная реплика.

**16. NoSQL облачные базы данных. Назначение, виды, особенности.**

Базы данных NoSQL предоставляют менее мощные инструменты запросов к данным по сравнению с реляционными базами данных, компенсируя это простотой разработки, более гибкой моделью данных и широкими возможностями горизонтального масштабирования и обеспечения высокой доступности.

Среди распространенных нереляционных моделей данных можно выделить:

**• Хранилища «ключ-значение» (примеры: Redis, Memcached)**

Самая простая модель данных. Использующие эту модель СУБД представляют данные в виде пар «ключ-значение». При этом поддерживаются операции занесения значения с ключом или получение значения по известному ключу. Некоторые СУБД поддерживают упорядочивание ключей и, как следствие, запросы по диапазону ключей.

Типичные варианты использования – высоконагруженные вебприложения, игровые сервера, управление сессиями, кэширование.

**• Документоориентированные базы данных (примеры: MongoDB,**

**Couchbase)**

СУБД, использующие данную модель, хранят структурированные данные с гибкой схемой, называемые документами.

В понятие и гибкости схемы входят следующие признаки:

1) документы не обязаны иметь одинаковый набор атрибутов;

2) типы данных атрибутов могут быть разными в разных документах;

3) значение атрибута может представлять собой массив;

4) значение атрибута может представлять собой вложенный документ.

Обычно документоориентированные СУБД позволяют индексировать данные (а значит, осуществлять поиск) по нескольким атрибутам документа.

Типичные варианты использования – Интернет вещей, системы управления контентом, каталоги, пользовательские профили.

**• Графовые базы данных (пример: Neo4j)**

Модель данных хранит данные, представленные в форме графа, включающего вершины и ребра между вершинами. Данные, удобно представляемые в такой модели, легко обрабатывать как графы и делать запросы к свойствам графа. Например, выполнять операции обхода графа, поиск в глубину или ширину, нахождения кратчайшего пути между вершинами.

Типичные варианты использования – социальные сети, аналитика, выявление мошенничества, управление сетевой инфраструктурой, рекомендательные системы, Интернет вещей.

• **Хранилища «семейство столбцов» (примеры: Apache Cassandra,**

**HBase)**

СУБД такого типа хранят данные в записях, которые могут содержать очень много (биллионы) столбцов. Имена и количество столбцов не фиксированы.

Типичные варианты использования – Интернет вещей, мессенджеры, обслуживание оборудования, логистика, оптимизация маршрутов.

*Большая часть крупных поставщиков облачных услуг предоставляет свои собственные решения NoSQL баз данных в виде сервисов, в том или ином виде совместимые с продуктами, упомянутыми выше.*

*Так, AWS предоставляет следующие сервисы:*

*• Amazon DynamoDB, «ключ-значение», документоориентированные базы данных;*

*• Amazon DocumentDB, документоориентированные базы данных (совместимость с MongoDB);*

*• Amazon ElastiCache, «ключ-значение» (совместимость с Redis,*

*Memcached);*

*• Amazon Keyspaces, «семейство столбцов» (совместимость сApache*

*Cassandra);*

*• Amazon Neptune, графовые базы данных.*

**17.Мониторинг и оптимизация затрат. Средства и особенности.**

Под мониторингом работоспособности системы в IТ понимают контроль параметров IТ-системы в заданных рамках, позволяющий предотвращать и оперативно устранять сбои в работе.

Для мониторинга любых IТ-систем, в том числе и облачных, применяются похожие технологии. Ввиду сложности современных IT-систем процесс контроля, как правило, разделен на несколько этапов, каждый из которых имеет свои особенности.8765

Средства:

NMS (Network Management Systems) («cистемы управления сетью») – программные и/или аппаратные средства, которые применяются для мониторинга и управления узлами сети.

NMS имеет возможность контролировать связи между устройствами, линиями связи в сети; сообщать об отказах на разных узлах сети. NMS собирает данные о состоянии узлов, трафике и представляет собой единую систему управления приложения, автоматизацию задач управления устройствами, обзор производительности, состояния сети, может выполнять сложные операции FCAPS (Fault, Configuration, Accounting, Performance & Security), применимые к сети в целом. В такой модели отражены ключевые функции администрирования сетей (управление отказами, конфигурацией, производительностью и безопасностью, а также учет). Стоит учитывать, что NMS не взаимодействует с сетевыми элементами на прямую, а управляет через EMS (Element Management System) («система управления элементами сети»), которая предназначена для управления и контроля отдельного сетевого элемента.

Protocol analyzers («анализаторы протоколов») – программные или аппаратно-программные системы, функцией которых является анализ, откладка, управление и мониторинг систем. Анализатор протоколов перехватывает пакеты и выполняет их декодирование. Процесс контроля работы сети, как правило, осуществляется в два этапа – мониторинг и анализ. Общая схема компонентов SNMP в сборе первичной информации о работе сети, такой как состояние портов концентраторов, коммутаторов, маршрутизаторов, количестве пакетов, а затем происходит анализ с сопоставлением данных и динамики их изменения. Анализатор протокола обычно выполнен в виде отдельного устройства или ноутбука со специализированным программным обеспечением.

APM-мониторинг

Рассмотрим наиболее типичные для облачных технологий аспекты мониторинга. Управление производительностью приложений (Application Perfomance Management, APM) – контроль производительности, предназначенный для мониторинга и управления приложения. Основная функция данного способа мониторинга – отслеживание производительности каждой транзакции приложения. Многоуровневый мониторинг приложений имеет широкий охват протокола связи, быстро выявляет сбои в производительности приложения с помощью мониторинга каждого компонента системы в отдельности, имеет свою комплексную оптимизацию производительности.

Мониторинг баз данных

Мониторинг баз данных подразумевает проверку их состояния для поддержания производительности и доступности. В настоящее время одной из частых проблем с производительностью приложений является замедление и неэффективность запросов к базе данных. Поэтому мониторинг их производительности является необходимым элементом системы контроля. Такой мониторинг подразумевает под собой проверку наличия свободного места на диске и оперативной памяти, целостность и доступность данных, контроль числа текущих подключений, времени выполнения SQL-запросов и прочие показатели. Также сюда входит проверка откатов, журналов на предмет ошибок, состояния периодического резервного копирования.

Мониторинг виртуальных машин

Поскольку виртуализация широко применяется в облачных системах, для них традиционный мониторинг обязательно включает в себя контроль облачными приложениями и управление выделением новых мощностей при росте нагрузки.

EUM-мониторинг

Мониторинг взаимодействия с конечным пользователем (End User Monitoring, EUM), или мониторинг пользовательского опыта – это показатель восприятия конечным пользователем каких либо задержек в сети, сбоев в работе приложения или сервиса. Инструменты EUM собирают данные о числе сбоев и времени выполнения пользовательских операций, предоставляя полную и подробную картину функционирования системы глазами пользователей. Являясь одним из направлений APM (Application Performance Management), мониторинг взаимодействия с конечным пользователем имеет наибольшую наглядность, предоставляя информацию о влиянии текущей архитектуры на эффективность выполнения задач конечных пользователей. Существует два типа EUM-мониторинга, один – синтетический мониторинг (работа по заданному сценарию) – выполняется независимо от того, пользуются приложением пользователи в данный момент или нет. Второй – так называемый мониторинг реальных пользователей – обеспечивает полное отображение реального текущего опыта работы пользователей с системой.

Оптимизация затрат:

Наряду с получением информации о системе и оперативным управлением ею в облачных системах часто используется анализ собранных данных для оптимизации работы (уменьшения времени простоя компонентов системы) и составления плана по расширению инфраструктуры. Для этого специализированное программное обеспечение на основе данных мониторинга составляет диаграммы наиболее частых последовательностей запуска приложений и их эффективности. Это позволяет понять, как пользователи взаимодействуют с системой и какие у нее узкие места. Также на основе типичных пользовательских сценариев создаются синтетические, запускающиеся по расписанию, сценарии для контроля успешности работы системы с точки зрения типичного пользователя. Таким образом, в конечном итоге системы мониторинга IТсервисов нацелены на формирование показателей степени доступности, а также качества предоставления сервисов.

**18.Шифрование в облачных системах. Подходы к обеспечению безопасности информации.**

Cloud Security («облачная безопасность») – направление в кибербезопасности, отвечающее за защиту данных, приложений и облачной архитектуры в целом от краж, утечек и удаления с помощью технологий, элементов управления и определенных наборов политик.

Шифрование данных – это кодирование информации при помощи ключа (пароля) с целью защиты данных определенного пользователя.

Методы шифрования делятся на:

• симметричные – используются для работы с информацией, где

применяется один и тот же ключ;

• асимметричные – состоят из пары ключей: закрытого и открытого.

Открытые ключи для асимметричных методов шифрования данных находятся в открытом доступе и используются для шифрования сообщений (узнать по нему закрытый ключ невозможно).

О закрытых знает только создатель; они используются для расшифровывания (если знаете закрытый ключ, можно подобрать и открытый).

Специфика шифрования данных в облаках

Для шифрования данных, хранящихся в облаках, применяются следующие подходы:

• использование специального тарифа облачного провайдера, обеспечивающего шифрование данных в облаке. Например, с использованием KMS (Key Management Service);

• использование специализированного ПО для шифрования «налету» данных, отправляемых в облако;

• самостоятельное шифрование данных перед помещением их в облако.

Эти подходы имеют свои особенности, делающие их применимыми в разных сценариях. Использование специального тарифа влечет дополнительные финансовые затраты. Специализированное ПО работает с отдельными облачными провайдерами, соответственно, при использовании разных провайдеров может возникнуть необходимость в применении разного ПО, что неудобно.

Шифрование в ручном режиме приводит к дополнительным затратам времени, хоть и может является в некоторых случаях более предпочтительным (данные передаются по сети уже в зашифрованном виде, их можно использовать с любыми облаками).

Также с последним подходом хорошо сочетается логика бэкапов – зашифрованные данные можно сохранять в локальном хранилище на случай потери данных в облаке.

Чаще всего в облачных технологиях используются следующие алгоритмы шифрования данных:

• ABE-шифрование;

• CP-ABE-шифрование;

• KP-ABR-шифрование;

• FHE-шифрование.

Шифрование с возможностью поиска

В облачных технологиях широко применяется класс криптографических алгоритмов шифрования, в которых есть возможность осуществлять поиск по зашифрованным данным (файлам, записям базы данных и т. д.). В общем случае в основе такого шифрования могут лежать как симметричные, так и асимметричные шифры. Однако, поскольку практические задачи предполагают шифрование большого количества данных, чаще встречается использование симметричного шифрования с поиском (Searchable Symmetric Encryption, SSE). Большинство асимметричных подходов основаны на шифровании по открытому ключу с поиском по ключевым словам (Public Key Encryption with Keyword Search, PEKS)

**19.Инфраструктура как код. Понятие. Сервисы AWS CloudFormation, HashiCorp Terraform.**

Определение:cture as Code») – это подход для управления и описания инфраструктуры через конфигурационные файлы, а не через ручное редактирование конфигураций на серверах. Основная идея IaC – описывать инфраструктуру кодом, переиспользуя практики из разработки ПО.

AWS CloudFormation:

Сервис AWS CloudFormation позволяет разработчикам и компаниям без труда создавать наборы связанных ресурсов AWS и сторонних поставщиков, обеспечивать их упорядоченное и предсказуемое выделение, а также управление ими.

Шаблон CloudFormation описывает требуемые ресурсы и их зависимости, что позволяет запускать и настраивать их вместе в виде стека. Вместо того, чтобы управлять ресурсами по отдельности, можно использовать шаблон для создания, обновления и удаления всего стека как единого целого.

Появляются 4 новых понятия:

• Шаблон – это декларированный файл кода JSON или YAML, который описывает надлежащее состояние ресурсов, необходимых для развертывания приложения.

• Стек реализует группу ресурсов, перечисленных в шаблоне, и управляет ей, а также позволяет управлять состоянием и зависимостями этих ресурсов совместно.

• Набор изменений – это ознакомительная версия изменений, которые будут выполнены операциями со стеком для создания, обновления или удаления ресурсов.

• Набор стеков — это группа стеков с совместным управлением, которые можно реплицировать как группу.

Шаблоны CloudFormation представляют собой текстовые файлы в формате JSON или YAML, состоящие из пяти типов элементов:

1. необязательный список параметров шаблона (входные значения, передаваемые в момент создания стека);

2. необязательный список выходных значений (например, полный URL адрес интернет приложения);

3. необязательный список таблиц данных, используемый для просмотра статических значений конфигурации (например, имен AMI);

4. список ресурсов AWS и их значений конфигурации;

5. номер версии формата файла шаблона.

HashiCorp Terraform

Terraform – ПО с открытым исходным кодом, используемое для управления внешними ресурсами. Создано и поддерживается компанией HashiCorp. Пользователи определяют и предоставляют инфраструктуру ЦОД с помощью языка конфигураций JSON.

Terraform управляет внешними ресурсами поставщиков услуг. HashiCorp поддерживает обширный список официальных поставщиков, а также может интегрироваться с поставщиками, разработанными сообществом.