

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

Институт цифровых интеллектуальных систем  
Кафедра компьютерных систем управления

Образовательная программа 09.04.01

«Информатика и вычислительная техника»

Дисциплина «Интернет-технологии.»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: |  |  |  |
| студент гр. ИДМ-22  -05 (ЦТ) | 31.10.2022  (дата) | \_\_\_\_\_\_\_\_\_\_\_  (подпись) | Кудратов Д.М |
| Принял: |  |  |  |
| преподаватель | \_\_\_\_\_\_\_\_\_  (дата) | \_\_\_\_\_\_\_\_\_\_\_  (подпись) | Овчинников П.Е.. |

Москва 2022

**Вопросы:** [**Билет 23**](https://github.com/stankin/inet-2022/wiki/exam23)

**1. Понятие доверенной среды. Способы обеспечения доверия в интернете.**

**2. Применение методологии и инструментов DevOps в веб-разработке.**

**Ответы**

**1.** Для успешного развития информатизации общества в "атмосфере" интенсивно растущих киберугроз с непредсказуемым изменением векторов атак необходимо создавать условия устойчивого и относительно безопасного удалённого электронного взаимодействия (УЭВ) с заданными характеристиками функциональной надёжности (ФН).

Информационные системы (ИС), спроектированные для работы в таких внешних условиях (критически-важные системы, дистанционное банковское обслуживание, здравоохранение, электронная коммерция и т.д.), в целях поддержания высокого уровня ФН, требуют создания так называемой доверенной среды (ДС).

Понятие ДС применительно к ИС пока не имеет устоявшегося определения, специалисты не могут сойтись во мнениях, а в нормативной базе также нет единого подхода, отделённого от контекста. Термин ДС появился в развитие понятия "доверенная система", введённого ещё в 1983г. в так называемой Оранжевой книге (TSSEC - Trusted Computer System Evaluation Criteria, критерии оценки доверенных компьютерных систем) – стандарте министерства обороны США [1], явившейся исходным материалом ряда руководящих документов Гостехкомиссии РФ, например, [2].

Так, под доверенной системой в стандарте [1] понимается система, использующая аппаратные и программные средства для обеспечения одновременной обработки информации разной категории секретности группой пользователей без нарушения прав доступа. Как трансформировалось это понятие в связи с эволюционным развитием вычислительных технологий? Что подразумевается под одним из самых дискуссионных понятий – "доверенная среда"? В данной статье попытаемся сформулировать некий подход к синтезированию ответов на эти вопросы на основе анализа ряда стандартов и научных работ.

**Что такое доверие применительно к результатам функционирования ИС?**

Дискуссия по рассматриваемому вопросу, развернувшаяся на круглом столе проходившего недавно IV форума АЗИ [3] показала, что начинать рассмотрение исследуемого вопроса следует с изучения самого понятия доверия. Рассмотрим, что такое доверие и как оно возникает.

Согласно стандарту [4] , разработанному для оценки доверия продукту (услуги) и его окружению, целью обеспечения доверия является создание уверенности в надёжном функционировании продукта в заданных условиях. Цель доверия определяется оценкой риска и/или политикой организации, при этом пользователь должен быть уведомлён об остаточном риске.

Все элементы безопасности, в частности, управление риском, независимо от применяемого метода, включают в себя неопределённость, которая исходит от множества факторов таких, как недостаточное знание всех аспектов функционирования, допусков при измерениях, экстраполяции величин и т.д. Значение этой неопределённости может быть настолько велико, что может являться основной составляющей остаточного риска. Сопутствующими факторами являются уязвимости эксплуатационной среды и несовершенство механизмов безопасности. При повышении уровня строгости доверия и усиления функций применяемых средств безопасности связанная с этими факторами неопределённость уменьшается, что сказывается на снижении величины остаточного риска.

Основываясь на этих положениях, попробуем связать возникновение доверия к выполнению определённого набора заданных функций ИС с её важнейшими эксплуатационными характеристиками:

* функциональной надёжностью, т.е. степенью готовности к выполнению заданных функций, пригодностью к восстановлению в случаях сбоев и ошибок, безотказностью, степенью достоверности результатов вычислений, контроля за процессом их получения [5];
* функциональной устойчивостью, т.е. способностью ИС противостоять деструктивным воздействиям [6];
* информационной безопасностью ИС, т.е. выполнению условий поддержания заданного уровня конфиденциальности, доступности и целостности информации, хранимой, передаваемой, принимаемой и обрабатываемой ИС в процессе её работы.

Будем считать, что применительно к ИС только выполнение в достаточной мере всех требований этих трёх составляющих, сбалансированных между собой по уровням их достижения, может приводить к определённому уровню доверия.

**Методы достижения доверия**

В международных стандартах [4, 7-11] представлено достаточно много методов обеспечения доверия. Приведём некоторые простые и понятные методы в качестве примера, при этом выделим вопросы, которые возникают при применении этих методов.

**Многократное повторение** определённого действия. Пример. Выдали банковскую карту. После двадцатого успешного снятия наличных денег в банкомате или оплате товаров в супермаркете появляется доверие к указанным процедурам. Если какое-то устройство эксплуатируется в течение длительного срока без сбоев и ошибок, успешно противостоит атакам, поддерживается весь его жизненный цикл, – оно вызывает доверие.

Однако могут быть скрытые отказы, недекладированные возможности (НДВ), которые могут вызвать тяжкие последствия. То есть, к многократному повторению необходимым дополнением должно являться знание архитектуры, компонентов и различных свойств устройства.

**Проверка (испытания).** Проверка соответствия стандартам и требованиям. Например, сертификация продукта или аттестация ИС. Заметим, что при этом возникает много вопросов о методике и полноте испытаний, а, главное, об ответственности организаций, проводивших испытания и подписавших итоговые документы.

Следовательно, эти вопросы могут быть транслированы в вопросы доверия к продукту, процессам и среде, в которой функционирует продукт, а, значит, и к самой информационной системе.

**Трансляция (перенос) доверия.**Доверие к самому источнику, который относит ИС, аппаратный или аппаратно-программный комплекс к доверенным или не доверенным. Естественными вопросами при этом будут являться полнота и степень корректности реализации методов и методик классификации уровня доверия.

В итоге можно лишний раз убедиться, что доверие не может быть абсолютным.

Всегда существует некая неопределённость, обусловленная постановкой задачи риск-менеджмента, которая может снижаться в процессе эксплуатации и постепенного раскрытия неопределённостей. Участие трёх перечисленных в предыдущем параграфе составляющих, имеющих вероятностную природу, диктует необходимость введения уровней доверия согласно определённым правилам. Эти правила можно будет сформулировать на основе требований, вытекающих из выполнения условий функциональной надёжности, устойчивости и информационной безопасности для ИС.

**Что такое компьютерная среда?**

Одним из дискуссионных вопросов является определение компьютерной среды (КС). По мере развития и совершенствования вычислительной техники этот вопрос становится все более неоднозначным. В терминах стандарта [4] понятие КС, переведённое с английского слова environment (строго говоря, окружение, окружающая среда, оборудование), в этом стандарте уже включает в себя все условия, в которых выполняются процессы жизненного цикла (т.е. люди, аппаратное, программное обеспечение и другие ресурсы). Такой подход впервые появился всего около 20 лет назад.

Вычислительные системы прошли довольно долгий путь развития и совершенствования. Компьютеры, а точнее, электронно-вычислительные машины (ЭВМ) примерно до 1990г. зачастую подразумевали управление вычислительным процессом (запуск и перезапуск ЭВМ, остановка и т.д.) с правами администратора многими из допущенных к вычислениям пользователей.

В те времена практически не было сетей, компьютеры стояли изолированно (stand alone). В то время не было компьютерных вирусов, вычислениям доверяли, но с осторожностью. Абсолютное большинство выявляемых ошибок составляли ошибки программирования. Однако надёжность ЭВМ была не столь высока, как сейчас. Сбои и ошибки в работе ЭВМ как электронного устройства были отнюдь не редки. Поэтому общее доверие к результатам вычислений ЭВМ было невысоким, все результаты вычислений подвергались сомнениям и обязательной проверке аналитическими, имитационными или экспериментальными методами. К тому, что стало пониматься как КС в те годы, как правило, относилась только сама ЭВМ.

Не всем пользователям нравилось ходить в вычислительный центр (ВЦ). Появление удалённых рабочих мест, связанных кабелем с ВЦ, постепенно перевело изолированный ВЦ в связанную чаще всего по схеме "звезда" вычислительную сеть.

С появлением компьютерных сетей к КС в зависимости от вычислительных задач стали относить либо само рабочее место, либо рабочую станцию, ВЦ и соединяющий их кабель, либо всю систему (не только все ЭВМ, но и связывающие их каналы).

Появление системы распределённых вычислительных систем, а в последние годы, и облачных вычислений, расширяет понятие КС до гигантских (в смысле физических параметров) размеров. Роль операторов при этом уже невозможно не учитывать. В то же время значительно актуализируется необходимость создания доверенной среды как такой части общей КС, которая гарантированно обеспечивает информационную и функциональную безопасность удалённого электронного взаимодействия и необходимых вычислений.

**Что такое доверенная среда? Как строятся доверенные системы?**

Определений ДС в научных статьях приводится много [12-17]. Все они разные, поскольку выработаны в разных контекстах. Нужен системный подход. Например, одним из вариантов такого подхода может быть принцип целеполагания: с какими целями создаётся и какие задачи решает ДС?

Представляет интерес также рассмотрение вопросов сложности создания системы доверия (к среде, продукту, процессам и собственно ИС) с помощью методов обеспечения доверия и строгости требований к доверию. Воспользуемся результатами стандарта [4], п.5.1, которые дополним на основе изложенного выше материала. В таблице 1 представлены полученные результаты анализа.

Таблица 1. Возможность использования методов обеспечения доверия.

Для иллюстрации приведенных в таблице результатов рассмотрим взаимосвязь и соотношение степени доверия к среде, процессу и продукту. Так, доверие к процессу согласно стандарту сосредоточено на процессах, применяемых к продукту на конкретных этапах жизненного цикла, в то время как доверие к среде сосредоточено на ресурсах и среде, в которой используются эти ресурсы. Уверенность в продукте, обеспечиваемая доверием к среде, обусловлена уверенностью в организации и её персонале, а также других ресурсах, взаимодействующих с продуктом. Такая уверенность может быть обеспечена аттестацией персонала или организации в целом.

При условии применения сопоставимой степени детализации доверие к среде обычно менее эффективно, чем доверие к процессу. Доверие к среде является самой низкой формой доверия и поэтому доверие к среде обеспечить легче всего.

В случае доверия к продукту утверждается, что характеристики продукта и его функционирования интенсивно тестировались и подтверждались в отношении корректности работы, пока не была получена требуемая степень доверия к продукту, являющаяся функцией применяемых критериев (что оценивалось) и методологией обеспечения доверия (как верифицировалось соответствие критериям).

Доверенная система строится на основе доверенных компонент, функционирующих в доверенной среде и реализующих доверенные процессы. С этой точки зрения согласно [18] доверие, – это свойство ИС, объективно, обоснованно и документально выраженное основание того, что элемент системы (в терминах стандартов – изделие ИТ, продукт ИТ, компонент ИС) отвечает априорно заданной (регламентациями высшего уровня) целевой функции на протяжении всего жизненного цикла.

Доверенные системы должны удовлетворять всем требованиям одного из, как минимум, трёх уровней безопасности (минимальный, стандартный и повышенный) и требованиям, зависящим от назначения ИС. Например, для систем дистанционного банковского обслуживания и интернет-банкинга нового поколения в [18] приводится 9 требований: к идентификации и аутентификации клиентов, идентификации и аутентификации клиента и удалённого банка, аутентификации и регистрации операций, защите транзакций, криптографической подсистеме, системе хранения ключей, безопасности программного окружения, журналам и аудиту, а также технологические требования.

**Доверенная среда в качестве базы доверенной системы**

Доверенная среда как основа создания и функционирования доверенной ИС может иметь как микроскопические размеры (например, чип), так и практически неограниченное физическими размерами пространство. В качестве одного из примеров доверенной среды в работе [19] рассмотрена банковская смарт-карта. Современная интеллектуальная смарт-карта - это целый маленький компьютер, состоящий из аппаратной части, операционной системы, системы управления картой, прикладного ПО, интерфейсов взаимодействия. Вопросы разработки архитектуры среды доверия подробно рассмотрены в главе 9 указанной работы.

Пример масштабируемой ДС – PKI (Public Key Infrastructure, инфраструктура открытых ключей). По сути это некая область доверия к открытым ключам. Правила простые: каждый отрытый ключ должен быть уникален и связан с конкретным владельцем. Ограничения: доверие распространяется только на вполне определённые функции. Пример попытки построения пространства доверия открытых ключей электронной подписи (ЭП) в Российской Федерации показывает, что этот процесс не решается только законодательными и подзаконными актами, а должен включать в себя набор согласованных нормативных, организационных и технических мер.

Однако использование инфраструктуры открытых ключей в качестве одного из видов масштабируемых ДС представляется весьма перспективным. Возможно расширение доверия на ряд сервисов. Принцип построения пространства доверия с определёнными уровнями применительно к сервисам идентификации и аутентификации приводится в работе [20]. Если исследованы и описаны все процессы и используемые продукты и они могут быть доверенными, то построение такого пространства возможно. Как показано в работе, при этом необходимо выработать согласованные требования к надёжности, устойчивости и безопасности используемых решений, что неизбежно влечёт за собой и уровни доверия к идентификации и аутентификации, а также выработку механизмов отнесения к определённому уровню доверия. Также показано, что необходимо создание института инспектирования как меры контроля за поддержанием среды, процессов, продуктов и компонент ИС в состоянии доверия заданного уровня. На базе PKI также может быть построено пространство доверия для обеспечения юридической силы электронным документам [21].

**Может ли ДС обеспечиваться исключительно техническими средствами?**

Виды доверенной среды: аппаратная, программно-аппаратная, с участием операторов (наличие человеческого фактора). Проще всего – аппаратная ДС. Однако в данном случае тоже много вопросов, вытекающих из рассмотренных выше разделов. Для аппаратно-программной ДС потребуются дополнительные исследования и выполнение критериев для достижения определённого уровня доверия.

Может ли быть доверенной ИС общего пользования? Условно может. Если предприняты меры по достижению определённого уровня надёжности, устойчивости, безопасности, и эти меры не скрываются, а предъявляются пользователям.

Вывод. Только техническими средствами ДС в подавляющем числе случаев недостижима. Кроме технических, как минимум, обязательно должны применяться оргмеры и/или меры обеспечения физической безопасности. Простым примером может служить тонкий часовой механизм в прочном корпусе.

**Киберугрозы и оценка рисков**

Лавинообразный рост количества различных киберугроз (вирусы, шпионское ПО, другие вредоносные программы, спам, фишинг, сбои, вызванные проникновением в сеть, DDoS – атаки и др.) вынуждает приоретизировать задачу противодействия им как весьма актуальную. Кроме перечисленных выше, существуют и непреднамеренные угрозы, исходящие от сетевых устройств. Преднамеренные, точнее, злонамеренные угрозы в виде заранее заложенных "бомб" с "часовым" механизмом, которые могут срабатывать уже без участия человека, например, троянов, стали по существу массовыми явлениями.

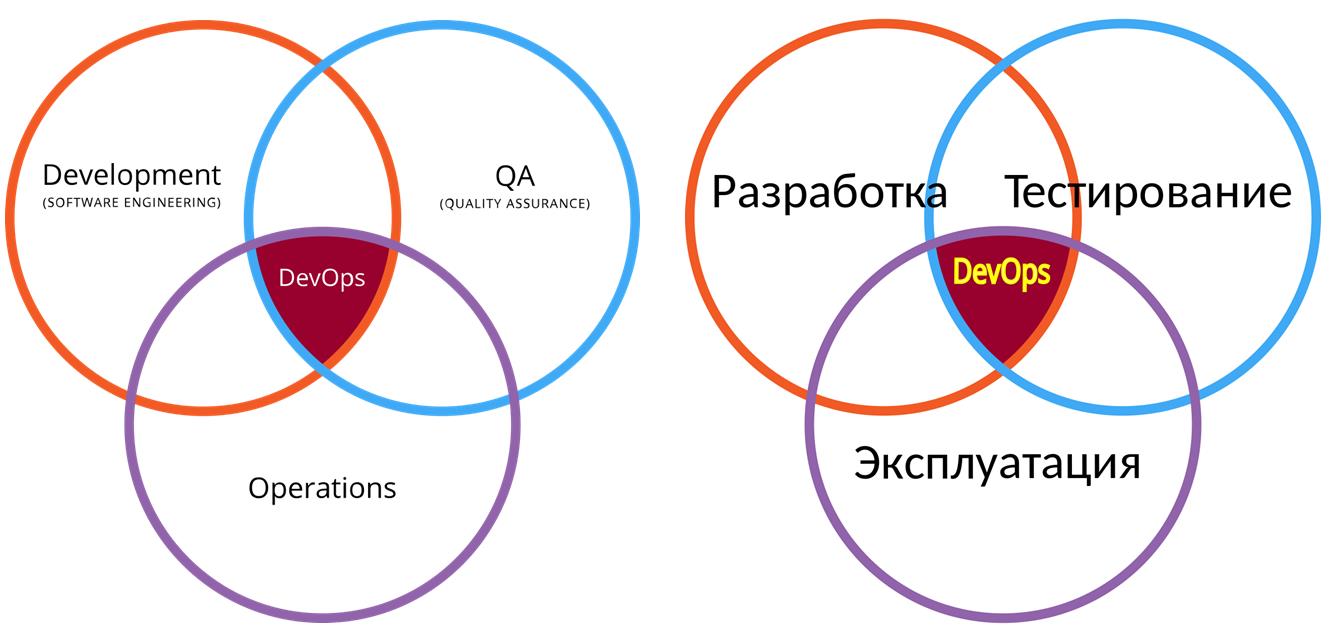
С точки зрения анализа рисков перечисленные угрозы и вероятность их реализации условно можно разделить на опасные события (хакерские атаки) и случайные события. По степени тяжести последствий (потенциального ущерба) иногда они могут быть сравнимы.

Задача противодействия, таким образом, сводится к поиску решений, способных снизить риски потенциального ущерба и вероятности его реализации до приемлемого уровня.



# **2. Методология DevOps и её влияние на облачные системы мониторинга**

В наше время сочетание культурной трансформации и автоматизации привело к переосмыслению идей того, как разработчики, сотрудники ИТ-подразделений и специалисты по информационной безопасности могут вместе работать. Ориентируясь на активное взаимодействие и интеграцию инженеров-программистов, тестировщиков и системных администраторов, методология DevOps базируется на идее о тесной взаимозависимости разработки и эксплуатации программных продуктов и сервисов, с целью их более быстрого создания и обновления. Но для того, чтобы такой подход стал успешным, жизненно необходимо осуществление всестороннего мониторинга в режиме реального времени.

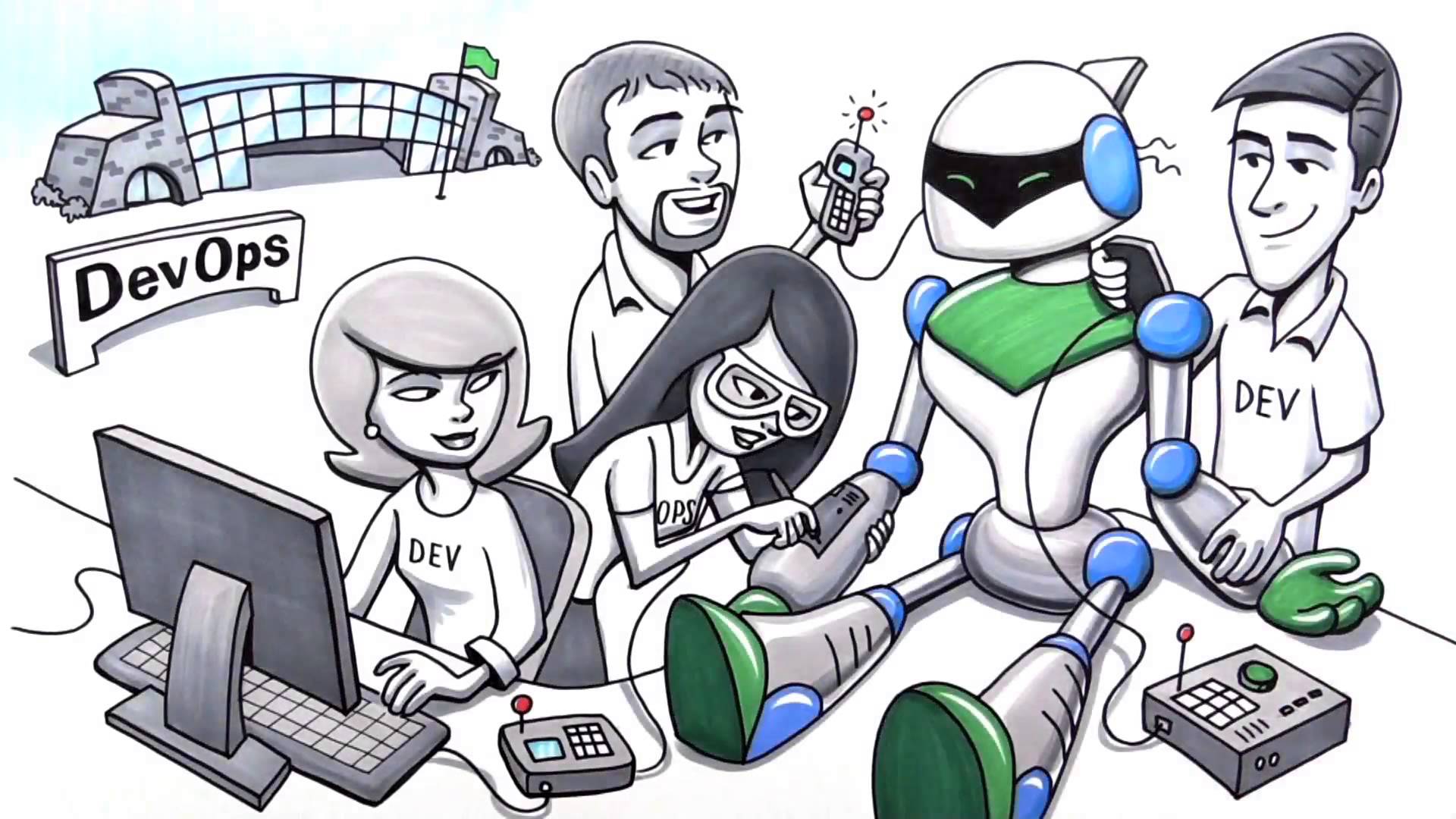


Методология DevOps в буквальном (акроним DevOps происходит от сочетания двух английских слов — development и operations) и образном смыслах представляет собой слияние двух ранее разъединенных процессов: разработки и эксплуатации программного продукта. В течение многих лет эти две группы были разъединены между собой границами, обусловленными как специфическими идеологическими различиями, так и наборами оперируемых знаний, что особенно было заметно в крупных ИТ-организациях корпоративного масштаба.

Это разделение было очень простым: фокус внимания разработчиков не распространялся дальше их программного кода, а системные администраторы хоть и работали с кодом, предоставленным им первой группой, но их задачей было взять этот код и убедиться в его работоспособности для конкретных условий. Полный разрыв между этими двумя группами обычно является основной причиной длительных циклов обеспечения качества (QA) и минимальным количеством производственных развертываний из-за опасений простоя или страха просто что-нибудь «испортить».

С относительно статической кодовой базой системным администраторам не нужны были наиболее чувствительные решения для мониторинга. Те же, что использовались, зачастую оперировали только с основными статистическими данными в производственной среде. При использовании такой каскадной методологии мог пройти год или даже больше между основными обновлениями программного обеспечения. Во многих организациях и по сей день продолжают придерживаться подобного подхода.

Тем не менее, в течение последних десяти лет произошел значительный сдвиг в подходе, который кардинально изменил весь ИТ-ландшафт. Разработчики, уставшие от постоянного ожидания обратной связи от производства для усовершенствования своего кода, начали писать программное обеспечение, которое автоматизирует операционные задачи. В то же время специалисты ИТ-подразделений начали делегировать свои глубокие знания и богатый опыт в программное обеспечение, написанное разработчиками.



Сегодня когда-то очень четкие границы между разработкой и эксплуатацией начали исчезать. Это приводит к ускорению всего жизненного цикла программного обеспечения, более коротким сессиям QA и многим другим изменениям. У нас даже появились новые процессы, такие как управление непрерывной интеграцией и непрерывным развертыванием. Многие большие программные решения, разработанные с помощью методики DevOps, развертываются по нескольку раз в день, а не несколько раз в год, как было раньше.

И это не какая-то новомодная фишка, которая скоро пройдет. Эта тенденция в автоматизации рабочих процессов прогрессировала постепенно в течение последних 10 лет, пока программное обеспечение и процессы не достигли своей зрелости. И влияние, которое DevOps уже оказывает и продолжит оказывать в будущем на информационные технологии предприятия, необходимо тщательно проанализировать с точки зрения всех вовлеченных участников этого процесса.

На данный момент осуществление мониторинга является одной из самых недооцененных областей при принятии методологии DevOps. Частая смена программного кода меняет подход к работе службы мониторинга, делая ежедневной нормой осуществление таких задач, как контроль передачи потоков в режиме реального времени, воспроизведение истории событий, продвинутая визуализация и т.д. Осуществление постоянного и качественного мониторинга становится критически важным компонентом всех приложений. Если вы не понимаете, почему мониторинг так важен для DevOps, вы, скорее всего, не готовы к тем изменениям, которые принесет с собой DevOps, а значит ваша компания останется в аутсайдерах на стремительно развивающемся конкурентном рынке.

**Какое влияние методология DevOps оказала на инженеров-разработчиков?**

Прежде чем мы углубимся в рассуждения о том, как и почему DevOps драматически изменит требования к современным решениям для мониторинга, давайте разберемся, как эта методология повлияла на разработку приложений. В конце концов, если вы не знакомы с инструментами разработки DevOps, вам сложно будет понять, почему подход к мониторингу полностью изменится.

Движение DevOps зародилось в 2009 году, но его истоки уходят еще глубже. Так, специалисты в качестве примера инструментария-прародителя называют популярное кроссплатформенное клиент-серверное приложение Puppet, первая версия которого появилась еще в 2005 году. Люку Канису (Luke Kanies), ныне основателю компании Puppet Labs и разработчику на Ruby в то время, надоело вручную настраивать Linux и вносить изменения в файлы конфигурации. Он мечтал о способе, который бы позволил предоставлять и настраивать Unix-подобные системы более программным и воспроизводимым способом. Таким образом, он написал скрипт на Ruby, который делал это вместо него, и назвал его Puppet.

Позже на рынке появился аналогичный инструментарий, включая Chef, Ansible, SaltStack и многие другие. Кроме того, вокруг этих инструментов образовались сообщества; разработчики и системные администраторы аккумулировали свои знания в виде «рецептов», позволяющих вам легко и быстро настроить программное обеспечение вне зависимости от используемого основного дистрибутива Unix.

С помощью подобного инструментария разработчики могут создавать самодостаточные программные сценарии того, как запустить приложение. Они могут включать все зависимости и запускаться на различных дистрибутивах Unix с помощью простого запуска скрипта. То, что раньше занимало несколько недель настройки в ручном режиме и выполнялось только высококвалифицированными специалистами, в настоящее время делается в считанные часы с помощью обычного скрипта.

Хотя разработчики и получили возможность разворачивать свой код быстрее и проще, чем раньше, они столкнулись с другой проблемой. Так как программисты стали меньше зависеть от своих коллег из операционного отдела, на разработчиков легла большая ответственность за работоспособность их собственных запущенных приложений.



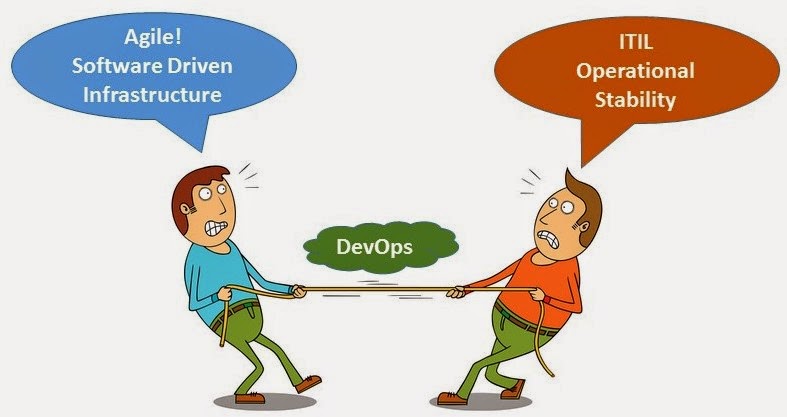
Эта проблема повлекла за собой появление на рынке еще одного инструментария, который впоследствии получил широкое распространение: модель предоставления облачных вычислений «платформа как услуга» (Platform as a Service, PaaS). Хотя первоначально данный инструментарий продвигался такими компаниями, как Salesforce и Google, первые реализации PaaS требовали от разработчиков писать специализированный код, который запирал приложения на их платформе. PaaS не был широко популяризирован, пока компания Heroku (в 2010 году поглощенная Salesforce) не представила одноименную облачную PaaS-платформу, поддерживающую ряд языков программирования. С ее помощью разработчики могли запускать свой код лишь с небольшим количеством существенных модификаций, если таковые вообще были необходимы.

Системы PaaS оказались на вершине принципов автоматизации DevOps. Более того, в наши дни большинство представленных на рынке PaaS-платформ используют DevOps-инструменты для настройки и запуска. Разница в том, что на PaaS-платформе приложения, которые запущены на ней, полностью управляемы. Вы можете запускать, останавливать, а также осуществлять масштабирование и мониторинг приложений в рамках PaaS-платформы через интерфейс программирования приложений (Application Programming Interface, API). В DevOps вы можете создать набор инструментов для управления вашими приложениями, а с PaaS вы получаете уже предварительно настроенный инструментарий.

И, наконец, сейчас ни одно из обсуждений DevOps не обходиться без упоминания Docker и Linux контейнеров. Пожалуй, наибольшим недостатком использования модели предоставления облачных вычислений PaaS является то, что она очень строго предопределяет архитектуру приложения. И если вы хотите получить больше контроля над средой выполнения, как раз Linux контейнеры вам ее предоставят, не повлияв при этом на скорость и гибкость выполнения кода. Создание среды с нуля с помощью инструментария Chef или Puppet может занять несколько часов, причем вы не сможете быть до конца уверенными в том, что в итоге вы работаете с точной ее копией. Использование же Linux контейнеров позволит разработчикам в считанные секунды воспроизвести среду Linux, при этом вы можете быть уверенными, что это ее точная копия.

Инженеры получили огромный выигрыш к своей производительности за счет многолетнего развития и улучшения инструментария DevOps. Дни разработчиков, не обращающих внимание на проблемы операционного функционирования и масштабирование своих приложений, подходят к концу, так как искусство запуска приложений неуклонно превращается в полностью автоматизированный компьютерный код.

**Какое влияние DevOps оказал на системных администраторов?**



Хотя разработчики и стояли у истоков революции DevOps, системные администраторы и специалисты по технологическим операциям стали ключевым звеном его популяризации. В конце концов, этот инструментарий помогает повышать эффективность и их работы, а не только программистов. В итоге использование инструментов DevOps трансформировало зону ответственности современной, гибкой команды специалистов по технологическим операциям и внесло кардинальные изменения в выполняемую ими работу.

До появления методологии DevOps системные администраторы и инженеры по технологическим операциям поддерживали отдельные приложения в большом масштабе. Это требовало выполнение таких задач, как настройка баз данных и веб-серверов, настройка балансировки нагрузки, управление безопасностью, управление системами кэширования и проведение многих других работ.

Инструментарий Devops обеспечил высокую степень стандартизации, предлагая эффективные способы развертывания, настройки и запуска многих серверов с помощью всего лишь нескольких автоматизированных инструментов, а не полагаясь на вмешательство операторов. Все чаще роль команды по технологическим операциям переориентируется на разворачивание и обслуживание автоматизированных приложений-сервисов, предоставляемых по требованию, к примеру, через модель PaaS или кластер контейнеров Linux. Разработчики развертывают и масштабируют индивидуальные приложения в сети устройств, оставляя команде системных администраторов запуск и масштабирование этих сетей.

DevOps платформы и инструменты создали буфер, который позволил командам разработчиков и специалистам по технологическим операциям работать независимо друг от друга, а не зависеть друг от друга. До появления этого инструментария существовал некий последовательный конвейер работы над программным обеспечением. Так, прежде чем программное обеспечение получалось окончательно развернуть, отдельные приложения для их использования нуждались в дополнительных индивидуальных настройках и доработках уже после осуществления закупки, а сервера необходимо было сконфигурировать.



С появлением инструментов DevOps разработчики получили определенные квоты, в рамках которых смогли разворачивать приложения по требованию и в режиме реального времени. Системным администраторам больше не нужно было беспокоится о разворачивании отдельных приложений. Да, команды по технологическим операциям по-прежнему осуществляли закупку оборудования, а также настраивали и управляли серверами, но делали они это на гораздо больших масштабах, чем уровень отдельного программного обеспечения. Их обязанность стала заключаться в управлении автоматизированными DevOps-сетями, к которым разработчики получили более свободный доступ.

Этот технологический буфер убрал строгую последовательность с жизненного цикла разработки и внедрения программного обеспечения, позволив разработчикам и системным администраторам работать более тесно друг с другом. На первый взгляд может показаться, что убирание тесной взаимосвязи между двумя командами для их большей интеграции противоречит здравому смыслу. Но вспомните, что самый лучший способ начать драку — поместить двух поваров на одной кухне. Позволив разработчикам работать в рамках четко определенной системы (например, построение кода внутри виртуальной сети устройств), в то время как команда специалистов по технологическим операциям работает вне контейнера Linux, гарантирует, что ваши повара получили и занимаются разными задачами.

Но золотой серединой между разработчиками и системными администраторами стала совместная экосистема (сами инструменты DevOps), где в наши дни стирается существовавшее ранее четкое разграничение обязанностей. Команды специалистов по технологическим операциям получают более глубокие знания о лучших практиках запуска и поддержки сложных программных систем. А разработчики, в свою очередь, получают больше возможностей для обучения компьютеров, внедряя более глубокие знания о реализации технологических процессов без вмешательства человека.

Поэтому не удивительно, что работодатели все чаще ищут инженеров с пониманием технологических операций, а системных администраторов с опытом в программировании. В целом же, однако, наибольшее влияние, которое DevOps оказывает на команды специалистов по технологическим операциям, заключается в том, что на системных администраторов все больше ложиться ответственность за запуск сети устройств, который обеспечит автоматическое развертывание программных опций по требованию, реализованных разработчиками.

**Как DevOps изменяет подход к мониторингу?**

Как вы, скорее всего, уже успели убедиться, методология DevOps во многом является современным эволюционным процессом, рожденным из старого способа ведения дел. Эта методология способствует появлению все большего количества автоматизированных процессов на всех уровнях жизненного цикла программного обеспечения, поэтому время выхода на рынок новых приложений значительно уменьшается. Тем ни менее, все эти изменения приводят к серьезным последствиям. Из-за тектонического сдвига в требованиях к разработке, тестированию и развертыванию программного обеспечения меняются нужды в функциональных возможностях современных облачных систем мониторинга.

DevOps ускоряет весь жизненный цикл программного обеспечения от разработки до процессов обеспечения качества. Относительно статическое ранее прикладное программное обеспечение сегодня может обновляться до нескольких раз в день. Это, в свою очередь, ставит перед разработчиками и системными администраторами множество задач, часть из которых старые, а часть — новые.

Разработчикам пришлось адаптироваться к новым условиям в том числе за счет написания более детальных автоматизированных тестов для своего кода, таким образом процессы QA становятся все более автоматизированными, насколько это вообще возможно. Обеспечение качества было поставлено в жесткую зависимость от непрерывной интеграции, которая автоматически запускает все программные элементы и интеграционные тесты всякий раз, когда передается новый код. Поэтому в настоящее время системы мониторинга становятся все более подготовленными к работе с каждой частью всего набора необходимых инструментальных средств DevOps.

До принятия методологии DevOps новые обновления программного обеспечения тщательно тестировались и принимались высококвалифицированными техническими специалистами. Непрерывное развертывание разительно поменяло принятый ранее подход. Сейчас процесс строится на максимальной автоматизации всего инструментария DevOps, и код вводится в работу всякий раз, как только он проходит все автоматизированные тесты.

Если такой подход представляется вам диким, и вы предполагаете, что его применяют в основном к наименее малым и незначительным приложениям, то вам стоит узнать, что, к примеру, компания Facebook уже долгое время является сторонником такого рода гибкого развертывания систем. Любой программист знает, что, если его код каким-то образом поломает Facebook, это будет отслежено к нему через историю управления версиями, и он понесет за это ответственность.

Все это очень напоминает историю-легенду о строителях мостов во времена Римской империи. Ее суть заключается в том, что, если ты строил мост в те времена, и он разрушался и убивал кого-то, тебя предавали смерти. Поэтому нет ничего удивительного в том, что часть этих мостов даже через две тысячи лет, прошедших со времени строительства, используется и в наши дни. Добавление личной ответственности обычно приводит к увеличению общего качества того, что вы строите.

Но многие организации не могут позволить себе слепо доверять некому черному ящику, который автоматически разворачивает код, работа которого с вашей стороны контролируется только на уровне надежды на опыт инженеров-разработчиков. Правильно реализованные системы мониторинга облачных сервисов могут обеспечить вам столь необходимое понимание всех процессов, помогая вам превратить автоматизированный программный Дикий Запад в нечто сродни Центру управления космическими полетами.

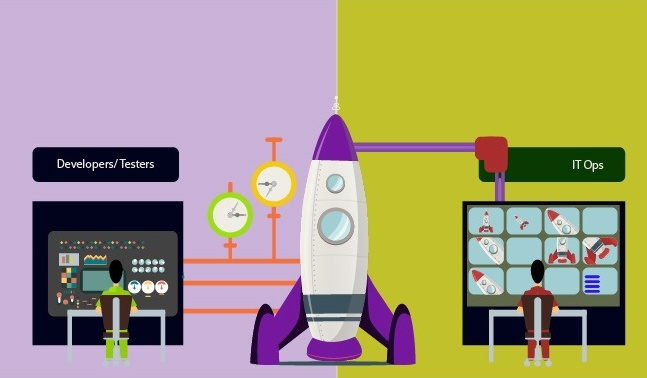
Современные системы мониторинга обеспечат вам большее понимание в режиме реального времени функционирования каждой части приложения, чем когда-либо ранее. Разработчики современного программного обеспечения пишут управляемый через интерфейсы API код, что привело к тому, что эти же интерфейсы программирования приложений сейчас стали доступны для систем мониторинга. Также многие сервисы мониторинга могут использовать специальные кодовые зацепки в самой логике программного обеспечения.

Кроме того, сервисы мониторинга расширили свое внимания от среды производственной эксплуатации до всего набора процессов выхода приложения. Они сейчас затрагивают этапы компиляции, тестирования модульных частей, интеграционные тесты, проверку того, как код выполняется при большой нагрузке и многое другое. К примеру, в компании Google сервисы мониторинга развертывания программного кода даже знают, как следить за программным обеспечением для управления проектами, высматривая и отмечая отдельные файлы, которые имеют статистически большее количество сообщений об ошибках во время разработки и тестирования, чем другие, обозначая их как горячие точки, чтобы следить за ними в будущем.

Надлежащий мониторинг в DevOps является активным, а не просто реактивным. Он находит способы улучшить качество ваших приложений, прежде чем проблемы даже дадут о себе знать. Так как такие системы мониторинга также следят и за набором инструментов DevOps, это может помочь улучшить этот инструментарий за счет выделения областей, которые, возможно, нуждаются в большей автоматизации.

Когда у вас есть сложное приложение, которое обновляется и разворачивается по нескольку раз на день и претерпевает быстрые циклы QA, вам бы хотелось бы идентифицировать проблемы настолько быстро, насколько это возможно. Осуществление сложного мониторинга становится первой линией защиты от простоев. Таким образом, системы мониторинга должны постоянно развиваться, чтобы принимать во внимание все новые данные. Это приводит нас к очень важному вопросу. В чем разница между сервисами мониторинга старой школы и сервисами мониторинга готовыми к DevOps? Если вы ошибетесь в выборе инструментария для осуществления мониторинга, вы столкнетесь с частыми и продолжительными простоями и упустите конкурентные преимущества от применения новых, гораздо более гибких технологий.

**Эволюция современных облачных систем мониторинга**



Очевидно, что многое изменилось в современном подходе к построению модели жизненного цикла программного обеспечения и его развертыванию, но многие поставщики решений для мониторинга все еще не успели адаптироваться к новым условиям и застряли в прошлом. На каждого вендора облачной системы мониторинга, готового к DevOps, существуют еще десяток, которые не отвечают всем требованиям современного рынка. Поэтому очень важно знать, на что обращать внимание, когда вы оцениваете представленные на рынке решения для мониторинга.

В современных DevOps-архитектурах комплексного программного обеспечения существует множество данных, которые необходимо отслеживать. Больше недостаточно отслеживать только самые простые статистические данные, такие как RAM, CPU и дисковый I/O. Теперь ваше решение для мониторинга должно работать с API и передавать данные непосредственно с самих приложений. Для того, чтобы понять всю эту информацию, одним из критериев поиска нужной вам современной системы мониторинга должны стать возможности анализа в реальном времени потоковых данных, воспроизведения истории изменений и еще лучшие средства визуализации.

Инструменты визуализации, в частности, имеют важное значение для целостного понимания состояния всех ваших приложений. Возможность выявлять проблемы в гибкой DevOps-среде зачастую зависит от качества ваших инструментов визуализации. Отслеживание проблем способом проверки отдельных лог-файлов, когда у вас появилось столь много изменяющихся частей, больше не является эффективной стратегией работы специалистов по технологическим операциям.

Следующим наиболее важным способом оценить облачную систему мониторинга является количество и качество поддерживаемых модульных интеграций. Работу со сколькими языками программирования поддерживает система мониторинга? Многие старые системы мониторинга работают лишь с несколькими, с особым фокусом на Java и .Net, несмотря на то, что высокоуровневые языки программирования становятся все более важными для разработки корпоративных приложений. Скорее всего, если не прямо сейчас, то в ближайшее время вы захотите себе новую систему мониторинга, которую можно будет завязать на популярные сценарные языки, такие как Python, Ruby, PHP и Go.



Позволяет ли программное обеспечение для мониторинга подключаться непосредственно к инструментам управления конфигурацией, таким как Puppet и Chef? Поддерживает ли оно общение с программным обеспечением, размещенным на PaaS-платформах, таким как Cloud Foundry и OpenShift? А как насчет Docker и Linux контейнеров?

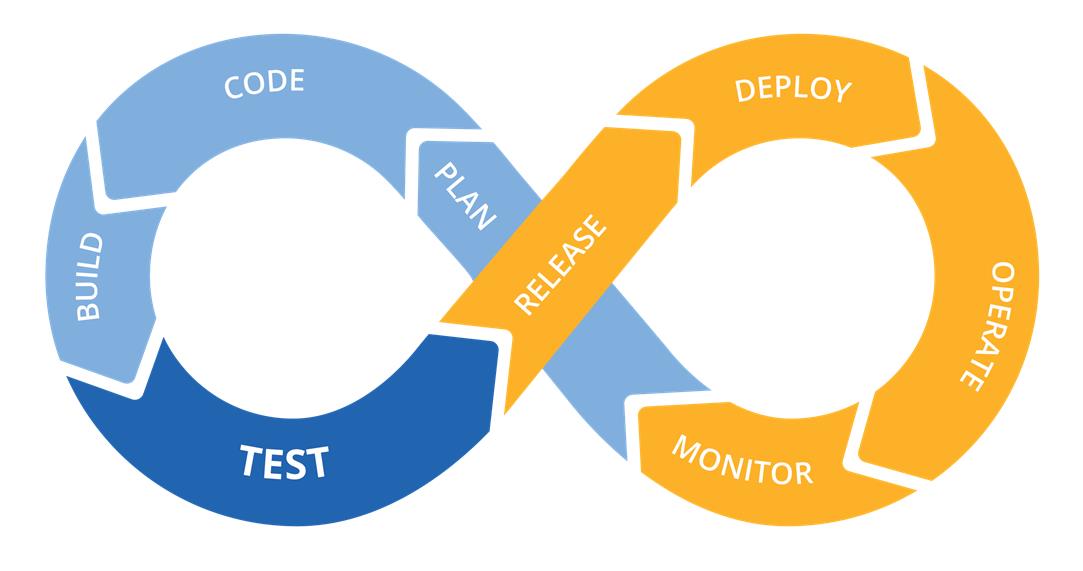
По сути, один взгляд на то, как программное обеспечение для мониторинга поддерживает работу с Docker контейнерами сразу скажет вам очень много о поддержке других современных популярных инструментов. Хотя программная платформа Docker и является одним из новейших инструментов DevOps, она уже успела застолбить за собой звание одного из наиболее быстро растущих сервисов с точки зрения принятия рынком.

В «тяжелом» сегменте программного обеспечения уровня предприятия до сих пор доминируют традиционные инструменты и инфраструктуры, такие как Java, .Net, Oracle, IIS, WebSphere и Microsoft SQL Server. Но предполагать, что облачным системам мониторинга достаточно будет поддерживать работу только с этими «старожилами» рынка, будет огромной ошибкой. Новые программные стеки все чаще «утяжеляются» с помощью Linux, PHP, Python, Ruby, Perl, Go, Nginx, Apache, Redis, Memcached, MySQL и PostgreSQL. Даже крупномасштабное программное обеспечение для автоматизации производственного процесса сейчас все чаще начинает использовать этот инструментарий. Как и крупнейшие мировые корпорации: Facebook, как известно, построен на PHP, Google часто и много использует Python и Go.

В обозримом будущем корпоративное программное обеспечение станет намного разнообразней, чем мы можем это наблюдать сегодня. Ведущие производители облачных систем мониторинга должны с умом подойти к этим изменениям, чтобы подготовиться к новым требованиям рынка. Чтобы не кануть в лету, им нужно охватить все грядущее многоязычное будущее и разработать инструменты, которые можно масштабировать со скоростью гибких DevOps-развертываний.

**Замкнутый круг**

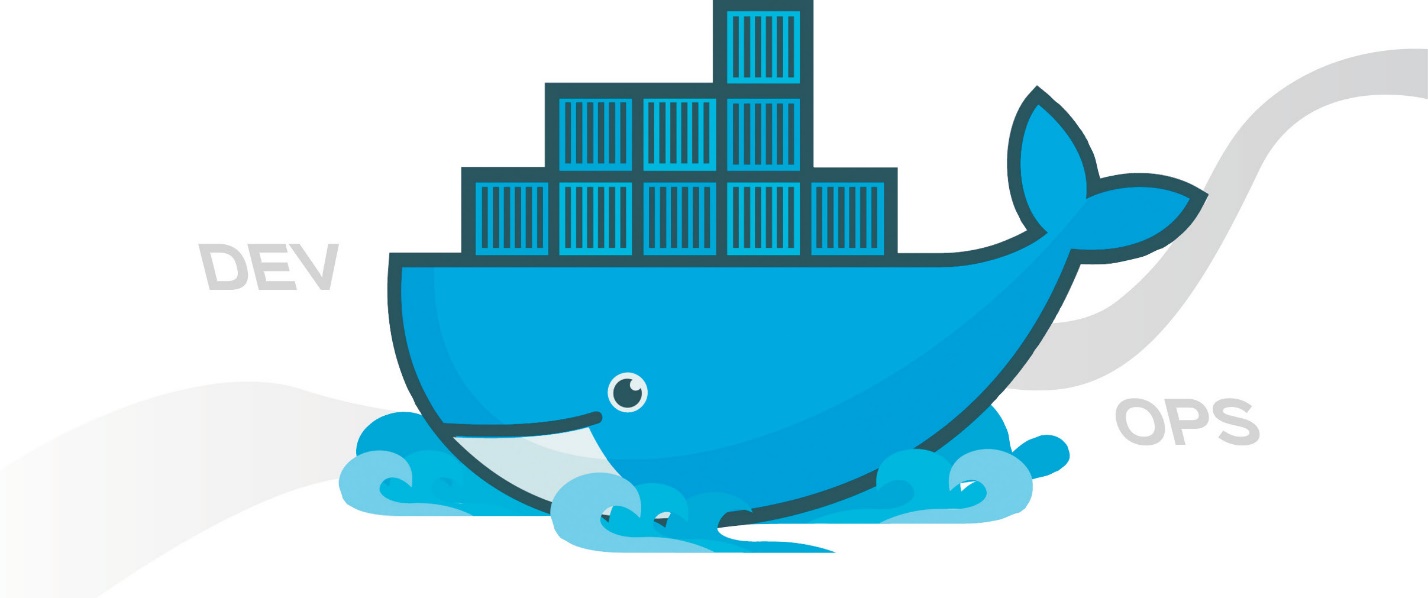
Знаменитые слова Марка Андриссена (Marc Andreessen) про то, что «программное обеспечение пожирает мир», одинаково справедливы для процессов запуска и мониторинга приложений, независимо от того, используются последние для вызова такси или бронирования мест в гостинице. Десятилетиями задача запуска программного обеспечения была привилегией элитной касты высококвалифицированных гениев — системных администраторов. Но сейчас это стало мейнстримом.



Раньше большая часть используемых для этого знаний не была задокументирована и передавалась в устной форме, как народный фольклор, из поколения в поколение. Все файлы конфигурации имели свои собственные эксцентричные форматы. Иногда эти форматы были настолько странные, что одни конфигурационные файлы компилировались в другие дополнительные конфигурационные файлы. Знания же, как именно и зачем настраивать эти файлы, оставались загадкой для всех, кроме нескольких избранных.

DevOps стал причиной тщательного документирования всех этих эзотерических операционных знаний в открытых стандартах, позволяющих записать и отследить все процессы и метрики в течение времени. Эти знания можно запрограммировать с помощью логики, используемой для построения сетки программного обеспечения на платформах, таких как PaaS и кластеры контейнеров Linux, и их можно повторять и совмещать с другими.

В то время как разработчики начинали все лучше разбираться в технологических операциях, а системные администраторы — в программировании, их обязанности, в конечном итоге, очень тесно переплелись. Границы окончательно размылись, и это открыло дорогу для появления нового инструментария для совместной работы между группами, одна из основных целей использования которого заключалась в минимизации времени взаимодействия всех вовлеченных в процесс групп.



Docker, лидирующая на рынке платформа на основе контейнеров Linux для автоматизации развертывания и управления приложениями в среде виртуализации на уровне операционной системы, является еще одним примером этой эволюции. GitHub, крупнейший веб-сервис для хостинга ИТ-проектов и их совместной разработки, предоставил разработчикам открытую среду для совместного использования кода друг друга и более легкого сотрудничества, чем когда-либо прежде. Облачный сервис Docker Hub только сейчас открывает подобного вида инструменты для совместной работы для системных администраторов, закладывая начало для нового подхода в управлении конфигурациями с открытым исходным кодом.

Инструменты для осуществления мониторинга должны принять во внимание все эти изменения. Они должны развиваться параллельно с техническими тенденциями и, в итоге, суметь удовлетворить потребности как разработчиков, так и системных администраторов, предоставив необходимую видимость для принятия новых технологий, таких как Docker, в производственную среду.

Из-за все более сжатых жизненных циклов программного обеспечения, наличие правильной системы для осуществления облачного мониторинга в режиме реального времени становится критически важным краеугольным камнем запуска и использования инструментов DevOps. Понимание всех изменяющихся частей и их взаимодействие между собой даст вам необходимую основу, чтобы решить, какое решение для мониторинга необходимо вашей организации.

Методология DevOps продолжит развиваться. Там всегда найдется место для новых инструментов, различных структур и крутых трендов, но основа, которая будет их все вместе связывать, заключается в выборе и правильном использовании программного обеспечения для автоматизации.