1 Вступление

Автор — Бенджамин Трейнор Слосс
Под редакцией Бетси Бейер

Надеяться — это плохая стратегия.

Традиционная поговорка SRE

Считается, что информационные системы не запускают себя сами — и это действительно так. Как же в таком случае *нужно* запускать системы — особенно большие и сложные?

Подход системного администратора к управлению сервисами

Исторически сложилось так, что для запуска сложных информационных систем компании нанимали системных администраторов («сисадминов»).

Подобный подход предполагает построение сервиса путем сборки существующих программных компонентов и их настройки для согласованной работы. Затем администраторам поручается запустить сервис и реагировать на происходящие события и появляющиеся обновления по мере их поступления. С ростом сложности системы и объема трафика количество событий и обновлений тоже растет, и команда администраторов также увеличивается, чтобы успевать выполнять всю эту дополнительную работу. Поскольку роль администратора требует набора навыков, который заметно отличается от навыков разработчика, эти два направления разделяют на две команды: команду разработчиков (development) и службу эксплуатации (operations, или просто ops).

Такая модель управления сервисами имеет ряд преимуществ. Для компаний, которые планируют запустить и обслуживать сервис, данный подход реализовать относительно легко. Поскольку эта парадигма существует довольно давно, есть много примеров, на которых можно чему-то научиться и которым можно подражать. Найти квалифицированных специалистов также не составляет труда. Множество существующих инструментов, программных компонентов (как готовых решений, так и заказных) и компаний-интеграторов помогут вам запустить такую систему, поэтому начинающим администраторам не придется изобретать велосипед и разрабатывать ее с нуля.

Однако подход с привлечением системных администраторов и разделением команд (dev и ops) имеет также несколько недостатков и подводных камней. Их можно разделить на две категории: явные и неявные издержки.

Неизбежные явные издержки достаточно велики. Функционирование сервиса, который требует ручных операций как при изменениях кода, так и при обслуживании событий, обходится все дороже по мере роста сложности сервиса и объема генерируемого им трафика, поскольку численность обслуживающей его команды также будет увеличиваться пропорционально трудоемкости.

Неявные издержки, сопровождающие разделение команд, не столь очевидны, но зачастую они гораздо дороже обходятся организации. Это проблемы, которые возникают из-за большой разницы в уровне знаний, наборе навыков и мотивации команд. Команды по-разному видят и описывают ситуацию; они делают разные предположения о рисках и возможностях реализации технических решений, о целевом уровне стабильности продукта. Разделенность команд может приводить к утрате не только общей мотивации, но и единства целей, взаимодействия между ними и в конечном итоге взаимного доверия и уважения, что будет иметь весьма болезненные последствия.

Традиционные службы эксплуатации и их коллеги-разработчики часто конфликтуют, особенно из-за сроков выпуска продуктов. Разработчики хотят быстрее запустить новый функционал и увидеть, что пользователи его приняли. Команда эксплуатации же хочет убедиться, что сервис не откажет в самый неподходящий момент. Поскольку бóльшая часть сбоев бывает спровоцирована какими-либо изменениями: новой конфигурацией, внедрением нового функционала или новым типом пользовательских запросов, — цели двух команд противоречат друг другу.

Обе команды понимают, что навязывать свои интересы друг другу неприемлемо («Мы хотим запускать все, что хотим и когда хотим, без промедления» против «Мы не хотим ничего менять в системе, если она работает»). И поскольку понятия, которыми они оперируют, и оценки рисков у обеих команд различны, зачастую начинается хорошо известная «окопная война», в которой каждый отстаивает свои интересы. Служба эксплуатации пытается оградить работающую систему от риска

изменений, связанных с введением новых функций. Например, ревизия новой версии перед запуском может подразумевать детальные проверки по *всем* проблемам, которые вызывали сбои в прошлом, — список может оказаться огромным, и не все его элементы будут одинаково значимы. Команда разработчиков быстро находит ответ. Они выпускают меньше «новых версий» и больше разнообразных «обновлений». Они начинают дробить продукт, чтобы на детальные проверки попадал меньший объем нового функционала.

Подход Google к управлению сервисами: техника обеспечения надежности сайтов

Но такие конфликты при выпуске программного обеспечения не следует принимать как нечто неизбежное. В компании Google был выбран другой подход: в наши команды SRE предпочитают набирать разработчиков, которые будут выпускать продукты и создавать вспомогательные системы для тех задач и функций, которые бы иначе выполняли, причем часто вручную, системные администраторы.

Что такое обеспечение надежности информационных систем (SRE) и как такой подход появился в Google? Я могу дать простое объяснение: SRE — это то, что происходит, когда вы просите программиста спроектировать команду службы эксплуатации. Когда я пришел в Google в 2003 году и передо мной поставили задачу возглавить группу эксплуатации «промышленных» систем в составе семи инженеров, я умел только разрабатывать ПО. Поэтому я создал группу такой, какой *мне* хотелось бы ее видеть, если бы я сам был SR-инженером. И управлял я ею соответственно. Эта группа в итоге выросла в современную команду SRE компании Google, и по сей день следующей своим принципам, заложенным человеком, который всю сознательную жизнь был инженером-программистом.

Главное в подходе Google к управлению сервисами — принцип формирования SRE-команд. Всех сотрудников SRE можно разделить на две основные категории: 50–60 % SR-инженеров — это разработчики Google или, если быть более точным, люди, которые были наняты по стандартной процедуре найма разработчиков Google; остальные 40–50 % — это те, кто имеет практически полную квалификацию разработчика (например, 85–99 % требуемых навыков) и *дополнительно* владеет навыками, полезными для SRE, которые редко встречаются у разработчиков. В данный момент мы чаще всего обращаем внимание на знание систем UNIX и работу с сетями (с первого по третий уровень модели OSI).

Всех сотрудников SRE объединяют их убеждения и стремление разрабатывать приложения для решения сложных задач. Внутри службы SRE мы тщательно отслеживаем развитие обеих категорий и на данный момент не видим разницы

в производительности между их представителями. На самом деле различия в квалификации членов SRE-команд часто позволяет создавать продуманные, высококачественные системы, которые, очевидно, являются результатом синтеза разнообразных навыков.

Такой подход к найму SR-инженеров дает команду, которая: а) быстро начинает скучать при выполнении задач вручную и б) имеет набор навыков, необходимый для создания программ, заменяющих ручную работу, даже если решение окажется сложным. В итоге SR-инженеры в основном выполняют ту же работу, которой ранее занималась служба эксплуатации, но их знания и навыки позволяют разрабатывать и реализовывать автоматизированные решения, заменяющие человеческий труд, и именно такую задачу ставит перед ними компания.

Изначально определено, что приоритетная задача для SRE-команд — именно разработка. Без постоянных доработок и улучшений в системе трудоемкость ее эксплуатации будет возрастать, и командам понадобится все больше людей только для того, чтобы успевать за ее ростом. В итоге для традиционной службы эксплуатации количество сотрудников растет линейно вместе с развитием сопровождаемого сервиса: если соответствующие продукты развиваются успешно, то с увеличением объема трафика увеличивается и нагрузка на группу сопровождения. Это означает, что придется нанимать все больше людей для выполнения одной и той же работы снова и снова.

Чтобы этого избежать, команда должна управлять потребностями системы, иначе она утонет в запросах. Поэтому в Google установлен *лимит 50 % для операционной работы всех SR-инженеров* — запросы на оперативное вмешательство в работу системы («тикеты»), дежурство, выполняемые вручную действия и т. д. Этот лимит гарантирует, что в расписании команды SRE достаточно времени на работы по улучшению сервиса, чтобы он оставался стабильным и работоспособным. Это верхняя граница. Со временем предоставленная сама себе команда SRE должна прийти к минимизации работ по эксплуатации и практически полностью посвящать себя разработке, поскольку сервис, по сути, работает автономно и восстанавливает себя сам: мы хотим, чтобы системы были *автоматическими*, а не только *автоматизированными*. На практике масштабирование и ввод нового функционала держит SR-инженеров в тонусе.

Итак, в Google есть простое ключевое правило: SR-инженеры должны тратить оставшиеся 50 % своего времени на разработку. Но как обеспечить такое распределение рабочего времени? Для начала мы должны узнать, как тратят свое время SR-инженеры. Имея эти данные, мы контролируем, чтобы команды, которые отдают разработке меньше 50 % времени, скорректировали текущий процесс. Зачастую это означает передачу некоторой части операционной нагрузки команде разработки или введение в SRE-команду людей без добавления дополнительных обязанностей по эксплуатации. Рациональное управление балансом между задачами по эксплуатации и разработке позволяет нам гарантировать, что SR-инженеры имеют

возможность создавать креативные и автономные инженерные решения, при этом сохраняя и знания, почерпнутые из опыта операционной работы.

Выяснилось, что основанный на SRE подход Google к построению и эксплуатации крупномасштабных систем имеет множество преимуществ. Поскольку SR-инженеры перерабатывают код, стремясь к тому, чтобы системы Google работали самостоятельно, для них свойственны стремление к быстрым инновациям и способность легко принимать нововведения. Такие команды относительно недороги — поддержка такого же сервиса только силами службы эксплуатации потребовала бы привлечения большего количества людей (операторов). Вместо этого количество SR-инженеров, достаточное для сопровождения и доработок системы, растет медленнее, чем сама система. Наконец, при таком подходе удается не только избежать проблем, связанных с размежеванием разработчиков и «операторов», но и повысить уровень самих разработчиков: без возможности легкого перемещения между командами разработки и SRE им было бы нелегко изучить особенности построения распределенной системы из миллионов узлов.

Несмотря на все эти преимущества, использование модели SRE связано с некоторыми трудностями. Одна из проблем, с которой постоянно сталкивается Google, — наем SR-инженеров: SRE и отдел разработки продуктов конкурируют за одних и тех же кандидатов. Кроме того, база кандидатов сравнительно невелика, так как в компании установлена высокая планка требований для навыков программирования и проектирования систем. Поскольку наша дисциплина относительно нова и уникальна, на текущий момент имеется не так уж много информации о том, как создать команду SRE и управлять ею (надеемся, что эта книга поможет исправить ситуацию!). Как только команда SRE будет укомплектована, ее потенциально непривычные подходы к управлению сервисами потребуют серьезной поддержки со стороны менеджмента. Например, решение приостановить выпуск версий до конца квартала лишь из-за того, что исчерпан лимит времени недоступности сервиса, может быть негативно встречено командой разработчиков, если только оно не санкционировано руководством.

DEVOPS ИЛИ SRE?

Термин DevOps появился в отрасли в конце 2008 года и на момент написания этой книги (начало 2016 года) все еще не вполне сформировался. Его основные принципы — привнесение IT-составляющей в каждую фазу проектирования и создания системы, максимальное использование автоматизации вместо человеческого труда, применение инженерных подходов и программных инструментов для решения задач эксплуатации — совпадают со многими принципами и рекомендациями SRE. DevOps можно рассматривать как обобщение некоторых основных принципов SRE для более широкого круга организаций, управленческих структур и персонала. В свою очередь, SRE можно рассматривать как конкретную реализацию DevOps с некоторыми специфическими расширениями.

Принципы SRE

Несмотря на то что особенности организации труда, приоритетов и ежедневных задач у разных команд различны, все эти команды имеют базовый набор обязанностей по отношению к сервису, который они обслуживают, и придерживаются одинаковых принципов. В общем случае команда SR-инженеров отвечает за *доступность*, *время отклика*, *производительность*, *эффективность*, *управление изменениями*, *мониторинг*, *реагирование в аварийных и критических ситуациях* и *планирование производительности* для своих сервисов. Мы систематизировали правила и принципы взаимодействия команд SR-инженеров с их окружением — не только с сопровождаемыми системами, но и с командами разработчиков и тестировщиков, пользователями и т. д. Эти правила и рекомендации помогают нам сосредоточиться на работе инженера, а не на операционных задачах.

В следующем разделе рассматриваются все основные принципы Google SRE.

Уделяем особое внимание инженерным задачам

Как мы уже говорили, по правилам Google на операционные задачи выделяется не более 50 % от общего времени SR-инженеров. Остальное время должно быть использовано для работы над проектами с применением навыков программирования. На практике это достигается путем наблюдения за количеством операционной работы, выполняемой SR-инженерами, и перенаправлением избытка таких задач командам разработчиков: переадресацией ошибок и поступающих запросов менеджерам по разработке, привлечением разработчиков к дежурствам и т. д. Перенаправление заканчивается, когда операционная нагрузка снижается до 50 % и менее. Это также обеспечивает эффективную обратную связь, ориентируя разработчиков на создание систем, не требующих вмешательства человека. Чтобы такой подход хорошо работал, все в компании — и SRE-отдел, и разработчики — должны понимать, почему существует это ограничение, и стремиться сократить объем генерируемой сопровождаемым продуктом операционной работы, дабы не провоцировать превышение лимита.

Занимаясь операционными задачами, каждый SR-инженер, как правило, получает не более двух событий за 8–12-часовую смену. Такой объем работы дает ему возможность быстро и точно обработать событие, привести все в порядок и восстановить сервис (в случае ошибки), а затем проанализировать причины произошедшего. Если появляется более двух событий за дежурство, проблемы обычно не удается изучить досконально и у инженеров не хватает времени для того, чтобы предотвратить будущие подобные ошибки, разобравшись в текущих. По мере масштабирования эта ситуация не исправляется. С другой стороны, если SR-инженер стабильно получает менее одного события за дежурство, его работа на месте дежурного — пустая трата времени.

Отчеты с анализом причин произошедшего (так называемые постмортемы) необходимо писать для всех значимых инцидентов, независимо от того, сопровождались ли они уведомлениями. Постмортемы для событий, уведомлений о которых не было, даже более ценны, поскольку они, вероятно, указывают на пробелы мониторинга. Подобное расследование должно установить все детали случившегося, найти все первоначальные причины проблемы и выработать план действий по ее устранению или улучшению способа обработки такого события в случае его повторного возникновения. В Google *никого не обвиняют в ошибках* — цель состоит в выявлении ошибок и исправлении их, а не в избегании или замалчивании.

Добиваемся максимальной скорости внедрения изменений без потери качества обслуживания

Команды разработчиков и SR-инженеров смогут наслаждаться эффективными рабочими отношениями, избавившись от противоречия между их целями. Это противоречие заключается в соотношении темпов внедрения изменений и стабильности продукта и, как говорилось ранее, часто проявляется неявно. В нашей модели SRE мы выводим этот конфликт на первый план, а затем избавляемся от него, вводя понятие *суммарного уровня*, или *бюджета, ошибок (error budget)*.

Это понятие основано на наблюдении, что *достижение 100%-ной надежности будет необоснованным требованием в большинстве ситуаций* (за исключением, например, кардиостимуляторов и антиблокировочных систем в тормозах). В общем случае для любого программного сервиса или системы 100 % — неверный ориентир для показателя надежности, поскольку ни один пользователь не сможет заметить разницу между 100%-ной и 99,999%-ной доступностью. Между пользователем и сервисом находится множество других систем (его ноутбук, домашний Wi-Fi, провайдер, энергосеть...), и все эти системы в совокупности доступны не в 99,999 % случаев, а гораздо реже. Поэтому разница между 99,999 % и 100 % теряется на фоне случайных факторов, обусловленных недоступностью других систем, и пользователь не получает никакой пользы от того, что мы потратили кучу сил, добавляя эту последнюю долю процента в доступность системы.

Если нам не нужно стремиться к 100%-ному уровню надежности системы, то к какому тогда? На самом деле это не технический вопрос — это вопрос к продукту, и вам нужно учесть следующие моменты.

- Какой показатель доступности удовлетворит пользователей, если мы знаем о том, как они используют продукт?
- Какие альтернативы имеют пользователи, не удовлетворенные доступностью продукта?
- Как отразится на активности обращений пользователей к продукту изменение уровня его доступности?

Продукт должен иметь установленный целевой показатель доступности. Как только этот показатель определен, допустимый суммарный уровень ошибок будет равен единице минус запланированный показатель доступности. Сервис, который доступен в 99,99 % случаев, недоступен в 0,01 % случаев. Этот допустимый уровень недоступности и есть не что иное, как допустимый *суммарный уровень ошибок (бюджет ошибок)*. Мы можем тратить этот «бюджет» на все, что сочтем нужным, пока не выходим за его рамки.

Как же мы собираемся потратить этот бюджет? Команда разработки намерена внедрять новый функционал и привлекать новых пользователей. В идеале мы могли бы потратить весь лимит количества возможных неполадок сервиса на риски, связанные с внедрением, чтобы быстрее запустить продукт. Это простое предположение характеризует в целом всю модель бюджета ошибок. Поскольку деятельность SR-инженеров строится в рамках этой концептуальной модели, высвобождение бюджета ошибок благодаря методам вроде поэтапного развертывания и одного «экспериментального» процента[1] — это оптимизация, которая позволяет ускорить процесс выпуска продуктов.

Применение принципа допустимого суммарного уровня ошибок разрешает противоречие интересов между командами разработчиков и SR-инженеров. Цель SR-инженеров уже не сводится к обеспечению отсутствия сбоев; вместо этого обе команды стремятся расходовать предоставленный бюджет ошибок так, чтобы максимально быстро внедрить новый функционал и выпустить продукт. Это и есть главное отличие. Сбои и дефекты (баги) больше не считаются чем-то «плохим» — это ожидаемая часть процесса внедрения новшеств[2]. Команды разработчиков и SR-инженеров теперь не боятся их, а управляют ими.

Мониторинг

Мониторинг (наблюдение) — это одно из основных средств отслеживания состояния системы и ее доступности. Мониторинг требует продуманной стратегии. Классический широко распространенный подход к мониторингу предусматривает наблюдение за определенным параметром или условием и, если заданное значение параметра превышено или условие выполнено, отправку оповещения по электронной почте. Однако такое оповещение по электронной почте — неэффективное решение: система, которая требует от человека прочесть электронное письмо и решить, нужно ли предпринимать какие-то действия, в принципе неполноценна. Система мониторинга никогда не должна требовать от человека истолковывать

[1] Здесь не вполне ясно, что конкретно имеется в виду. Скорее всего, ограничение внедрений опытных версий и проведения экспериментов над действующими системами одним процентом от общего их количества. — *Примеч. пер.*

[2] Разумеется, и авторы упоминают это, такой подход возможен лишь при условии, что ошибки и сбои действительно допустимы. — *Примеч. пер.*

какую-либо часть оповещения. Вместо этого всю интерпретацию должно выполнять программное обеспечение, а люди будут оповещены только в том случае, когда от них требуется предпринять какие-либо действия.

Существует три категории данных от системы мониторинга.

- *Срочные оповещения* (*alerts*, *«алерты»*) — указывают, что нужно немедленно реагировать на что-то, что либо уже произошло, либо вот-вот произойдет.
- *Запросы на действия* (*tickets*, *«тикеты»*) — указывают, что человеку нужно вмешаться, но не обязательно немедленно. Система не может обработать ситуацию автоматически, но предпринимать какие-то действия допустимо не сразу, а в течение нескольких дней без каких-либо негативных последствий.
- *Журналирование (logging)* — нет необходимости кому-либо просматривать эту информацию, она записывается для диагностических целей или для последующего анализа. По умолчанию журнал читать не требуется, пока что-либо иное не заставит сделать это.

Реагирование на критические ситуации

Надежность — это функция от среднего времени безотказной работы (mean time to failure, MTTF) и среднего времени восстановления (mean time to repair, MTTR) [Schwartz, 2015]. Наиболее значимый критерий при оценке эффективности реагирования на аварии и другие критические ситуации — это быстрота восстановления работоспособности системы, то есть MTTR.

Выполнение операций вручную приводит к увеличению задержек. Система, способная избегать аварий, которые потребовали бы ручного вмешательства (хотя бы в отношении только наиболее частых сбоев), будет иметь лучшие показатели доступности, чем если бы она нуждалась в таком вмешательстве всегда. Рассматривая ситуации, когда вмешательство людей все же необходимо, мы обнаружили, что продумывание всех деталей и превентивная запись методических рекомендаций в инструкцию приводит к практически трехкратному улучшению времени восстановления (MTTR) по сравнению с неподготовленными импровизациями. Конечно, героический дежурный инженер, мастер на все руки, выполнит всю необходимую работу, но обычный опытный дежурный инженер, вооруженный инструкцией, справится с этим гораздо лучше. Несмотря на то что ни одна инструкция, какой бы полной она ни была, не заменит толковых инженеров, способных импровизировать на ходу, четкое и исчерпывающее описание шагов и советы по поиску неисправностей очень ценны в тех ситуациях, когда нужно отреагировать на критическое или не терпящее промедления происшествие. Поэтому в Google SRE при подготовке дежурных инженеров полагаются на инструкции, в дополнение к упражнениям вроде «Колеса неудачи»[1].

[1] См. подраздел «Катастрофа: ролевая игра» раздела «Пять приемов для вдохновления дежурных работников» главы 28.