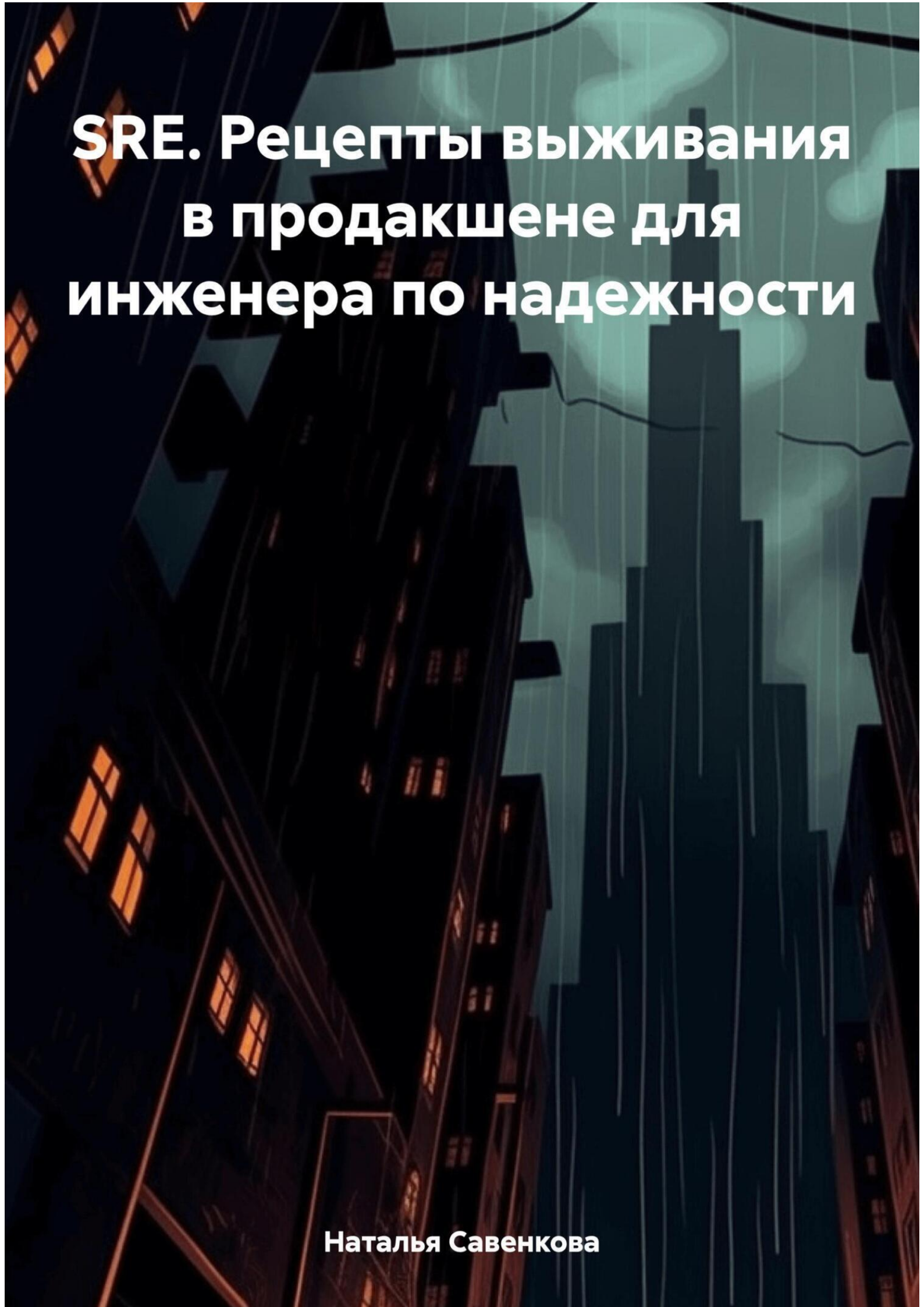

SRE. Рецепты выживания в продакшене для инженера по надежности

Наталья Савенкова

Наталья Савенкова

SRE. Рецепты выживания в продакшене для инженера по надежности

«Автор»
2024

Савенкова Н.

SRE. Рецепты выживания в продакшене для инженера по надежности / Н. Савенкова — «Автор», 2024

Мир IT меняется довольно быстро, но внутри остаются всё те же сервера, каналы, базы данных и пользователи. В книге собраны простые и полезные рецепты для жизни инженера по надёжности, описан алгоритм создания инцидент-менеджмента в компании.Основано на реальных событиях и собственном опыте.

Содержание

Наталья Савенкова
SRE. Рецепты выживания в продакшене для инженера по надежности

Что внутри

С теплыми чувствами к моим коллегам из чата сарказма и котиков.

Здравствуй, читатель! Я Наташа и я инженер. Двадцать лет я работаю в IT, и мой путь начинался, как у многих инженеров того времени, с веб-мастера, а интернет тогда работал по телефонному проводу. Моя история опыта в индустрии крутится в основном вокруг бекенда и инфраструктуры.

На своей первой серьезной работе мы делали интернет-магазины, поэтому понятие надежности систем довольно быстро вошло в мою жизнь: если интернет-магазин не работает, то компания не может обслуживать заказы, а у его владельца прекращается поток денег. Для таких бизнесов IT-система – это в прямом смысле сердце. С тех пор мир поменялся очень круто и такое электронное сердце теперь есть, пожалуй, у всех.

В 2015 году я пришла работать разработчиком в крупную компанию и там стало очень быстро понятно: если у такой компании не работает ее главный сайт, то об этом сразу пишут в новостях. Это очень смешанные чувства: ответственность и гордость одновременно. В мире начал набирать популярность подход “Site Reliability Engineering”, в наш отдел в компании добавили админов, которые сели за соседний со мной стол… и надежность стала моим главным профессиональным интересом.

Что нужно знать о надежности:

– это не бесплатно

– это про готовность заниматься системой в любой момент

– это для педантичных

– это про постоянное извлечение уроков и изучение ошибок

Мир IT как будто меняется очень быстро, но фундаментально за 20 лет мало что изменилось: новые языки программирования каждый год, облачные технологии, serverless, zero-code, ML, базы данных и еще много всего нового, но внутри все те же сервера с процессорами, каналы связи, дата-центры и экскаваторы, которые неловким движением перерубают кабели в земле.

В этой книге собраны мои правила и рецепты, накопленные за все время работы инженером по надежности. Если для рецепта будет актуально, то буду добавлять в него что-то про деньги. В конце концов, мы делаем IT-систему для бизнеса, а бизнес всегда про деньги.

Рецепты в основном для крупных систем, но и для небольших тоже что-то будет полезно. Никакой логики в порядке глав тут нет. В книге много сленга и она рассчитана на инженеров с опытом работы.

В конце книги будет глава с пошаговым планом по созданию процесса "инцидент-менеджмент" в своей компании.

Основано на реальных событиях. Приятного чтения!

1. Сервис без вмешательства не переживает отключение части свитчей в дата-центре – это плохой сервис

Пришли к нам как-то сетевые инженеры из дата-центра и говорят: "нам нужно провести работы, для этого мы выключим пару свитчей, запланируйте у себя мероприятия". Обычно в таких ситуациях мы начинали какой-то трафик куда-то переключать, чтобы точно все хорошо прошло, а тут пообсуждали с коллегами и решили, что это неправильная ситуация и лучше мы посмотрим на последствия, а потом что-то улучшим. Всю систему оставили работать в обычном режиме, подготовились к "чему угодно" и стали наблюдать. Все прошло хорошо. С тех пор мы договорились, что на такие работы ничего сами трогать не будем, потому что система должна суметь сама.

Деньги: если система сама не сумела, то нужно оценить масштаб последствий для бизнеса, оценить варианты улучшения системы и принять решение об инвестициях в улучшение системы. Допустимо оставить как есть, если улучшения будут стоить неоправданно дорого.

2. Если какую-то процедуру делать страшно – делай ее чаще

У каждого инженера по надежности или администратора системы есть набор нелюбимых манипуляций в системе, которые делать страшно, но все равно иногда приходится. Я выработала для себя правило: если у меня есть такая процедура, то мне самой нужно ее просто регулярно повторять, чтобы она становилась привычной.

Почему это важно. У каждого из нас разная реакция на стресс: бей-беги-замри. Когда что-то сломалось, то запускается стресс. Когда нужно во время этого сломанного провести нелюбимую манипуляцию, то стресс увеличивается еще больше.

Как-то в моем хозяйстве была кластеризованная база данных. В работу базы вообще вмешиваться неуютно, но иногда (редко) надо было отключать некоторые из ее нод. Очень неприятная процедура. Я завела себе плановые работы раз в месяц по отключению нод: проверяла, что оно правильно работает, а заодно и повышала свой комфорт от процедуры.

3. Если мониторинг не пишет о проблемах – проверь, возможно он не работает вообще

На серверах лежат файлы, а у файлов есть права доступа. Мониторинг часто устроен так, что просто читает заданные файлы с логами.

Как-то мы переезжали с одних серверов на другие, и что-то пошло не так с правами доступа на файлы логов сервиса бекенда. В результате на некоторых серверах бекенд не мог писать свои логи. Нет логов – нет проблем. Мониторинг читал пустые файлы, не находил там никакой тревожной информации и всегда показывал "все в порядке". В это время на машинке оставался необновляемый код, а пользователь, попадающий запросами на эти сервера, видел вообще нечто очень странное. Нашли мы это случайно, к сожалению.

Отсюда следует: если мониторинг настроен по правилу "нет ошибок – нет проблем", то его стоит дополнить проверками, показывающими, что система действительно работает как задумано.

4. Регулярно проверяй все редко используемые аварийные средства доступа

В работе ответственного админа есть не только основные рабочие средства, но и резервные средства. Резервный интернет, резервный ноутбук, еще разные запасные способы, типа возможности залогиниться на сервер с телефона или загрузочной флешки.

Если запасными средствами не пользоваться, то рано или поздно они перестанут работать. Такова судьба запасных средств. Поэтому важно регулярно проверять, что эти запасные средства до сих пор функционируют и могут быть использованы в критической ситуации.

Например, можно сделать себе напоминание раз в две недели "проверить резервные средства" и там описать все, что нужно проверить: резервный интернет оплачен и работает, резервный ноутбук загружается и с него можно зайти во все необходимые системы и так далее.

5. Ходить на чужие разборы полезно

Во многих компаниях есть процесс публичного разбора крупных инцидентов (поломок). Это прекрасная практика, хотя и малоприятная для самих выступающих и участников инцидентов. Задача публичного разбора – сгенерировать с помощью большого числа инженеров меры предотвращения подобных поломок в будущем.

Если у вас в компании есть такое мероприятие – ходите туда и учитесь на кейсах своих коллег. Не надо ждать, когда случится инцидент именно у вас. Уникальных проблем по истине мало, а способов их предотвращения еще меньше. Изучайте, что случилось у коллег, анализируйте свою систему и выбирайте то, что разумно заранее реализовать в вашей системе.

Если такого процесса не существует, то подумайте о том, чтобы он появился. Вариант реализации такого процесса будет описан подробнее в последних главах этой книги.

6. Если результаты нагрузочного тестирования всегда одинаковые – это плохо

Если вы уже выкатываете релизы автоматически и в процессе выкатки есть стадия нагрузочного тестирования, то этот рецепт для вас.

В нашем релизном процессе был шаг выкатки на тестовый стенд, на который выкатывается сборка и нагружается трафиком. Чтобы не задерживать сильно релизный процесс, мы выставили довольно высокое стартовое значение нагрузки по принципу "ну, столько наш бекенд точно выдержит всегда". Затем система тестирования плавно увеличивала трафик. По мере повышения трафика стенд переставал отвечать на запросы, тестирование завершалось, а последнее успешное значение трафика принималось за результат нагрузочного тестирования. Если результат был допустим, то релиз выкатывался дальше в продакшн.

Долгое время наш результат тестирования был более менее стабильным. Потом добавили немного логики, потом еще немного, потом еще немного… А результат продолжал оставаться стабильным и релизы выкатывались в продакшн. Пока кто-то не пошел зачем-то посмотреть результаты тестирования своими глазами…

Что произошло на самом деле: по мере добавления новой функциональности и деградации производительности уровень допустимого трафика на стенд постепенно падал и упал ниже заданного стартового значения. В итоге, тестирование заканчивалось сразу же, как только начиналось, потому что стенд обслуживал несколько запросов и сразу же отваливался, а в результаты просто записывалось то самое стартовое значение. За это время производительность бекенда упала на 50%, но об этом никто не знал.

Как стоило бы сделать:

– начинать нагрузку трафиком с нулевого значения, но это сильно замедляет процесс релиза

– сделать параллельный процесс полного нагрузочного тестирования, чтобы не задерживать релизы

– считать тестирование успешным в случае, если финальное значение отличается от стартового

– считать долю успешных и неуспешных ответов от стенда

7. Регулярно проверяй всю редко используемую автоматику

Одним из основных принципов SRE является проактивное управление системами, что означает создание автоматических систем для защиты от инцидентов и поломок разного рода.

Вот несколько примеров таких автоматик:

– включение фильтрации трафика при срабатывании каких-то условий
– автоскейлинг ресурсов при росте нагрузки
– подключение кеширующих прокси
– отключение незначимых компонентов системы при пиковой нагрузке
– снижение скорости передачи данных
– увеличение времени ответа
– …

Список вариантов большой, но смысл понятен.

Что важно: речь идёт о автоматике, включающейся при некоторых условиях. Это означает в свою очередь, что это редкие ситуации. И это же означает, что механизмы должны работать безотказно. Как огнетушитель в вашем деревянном загородном доме с дровяной печью: если случится так, что он пригодится, то лучше будет, если он будет исправен.

Всю такую автоматику необходимо регулярно проверять! Сделайте себе расписание учений и протоколы проверки всех автоматик, на которые вы полагаетесь для обеспечения высокого качества своего сервиса в критических ситуациях.

В ходе этих регулярных проверок вы сможете обнаружить:

– Изъяны или слабые места до того, как они проявятся в результате реальных инцидентов.
– Изменения окружающей среды: по мере развития сервисов и инфраструктуры защитные механизмы могут потребовать корректировки или вообще перестать работать.
– Несоответствия требованиям аудита
– Неполадки в работе системы мониторинга и оповещений
– Отсутствие необходимых доступов
– … и ещё много всего.

Кроме того, участие в тестировании автоматики это хороший способ онбординга новичков в команде.

Каждая проверка – это возможность узнать больше о системе и о том, как она ведет себя в различных условиях, что в итоге приводит к совершенствованию защитных механизмов.

Деньги: тут крайне важно соблюдать баланс между “давайте подготовимся заранее к чему угодно и будем оберегать наш хрустальный дворец” и “ничего не делаем вообще”. Если вы не делаете систему жизнеобеспечения, не управляете ракетами и прочими критическими системами, то будет достаточно:

– проанализировать систему на предмет основных рисков
– оценить потери в результате реализации рисков
– спроектировать средства защиты
– оценить стоимость их реализации и поддержки
– применить здравый смысл и выбрать, куда потратить свои деньги

8. Рандомизируй учения

В прошлой главе было много слов про важность проверки систем и протоколы проверок. Назовём эти проверки учениями.

У любых учений есть один главный недостаток: они далеки от реальной катастрофы. И второй недостаток: они проводятся по протоколу.

К сожалению, если на учениях выявилась какая-то проблема у какого-то сервиса, то устранение этой проблемы означает только то, что сервис научился переживать сценарий учений. Это вовсе не значит, что если начать отключать что-то в другом порядке, то всё будет хорошо. И уж тем более не значит, что авария будет проходить по сценарию учений.

Вносите разнообразие в учения. Регулярно меняйте протоколы и форматы.

Изменение последовательности действий во время учений повышает шансы того, что отдельные люди и команды действительно понимают лежащие в основе принципы и готовы реагировать на неожиданные ситуации.

Вот несколько способов внести разнообразие:

– Использовать генератор случайных чисел, где это применимо

– Использовать временные вариации: менять время проведения учебных проверок, например, проводить их в разное время суток

– Вместо одного сценария, представить варианты, когда различные компоненты выходят из строя в разном порядке или возникают несколько проблем одновременно

– Замена ролей: менять членов команды ролями во время учений, это не только изменит динамику, но и покажет проблемы в навыках

– Изменение последовательности: менять порядок шагов в сценарии учений, чтобы увидеть, как участники адаптируются, смогут ли они по-прежнему эффективно решать возникающие проблемы, и как будет меняться поведение всей системы

9. Проектируй failover смолоду

Если у сервиса есть хоть какой-то шанс получить статус "должен работать примерно всегда", то лучше на это закладываться пораньше. Сами процессы стоит проектировать реентерабельными – рассчитанными на перезапуск, параллельный запуск и какой угодно другой запуск и работу. Лучше сразу предполагать, что любая часть проекта может выйти из строя, и резервировать её, если без неё нельзя обойтись. Во-первых, оно будет более-менее устойчивым, а во-вторых более горизонтально масштабируемым.

Сделайте визуальную схему всей системы и спроектируйте меры повышения надежности.

Деньги: резервирование системы увеличивает стоимость системы не в два раза, а существенно больше, так как для управления резервными схемами требуются инструменты координации.

Как и в случае с рецептом про автоматику, здесь стоит оценить последствия отказа конкретных компонентов, посчитать стоимость резервирования компонентов, а также стоимость систем координации. Только после этого принимать решение о создании резервирования.

10. Мониторинг трафика в диапазоне

Каждый раз, когда пользователь взаимодействует с приложением / веб-сайтом, отправляя данные или выполняя поиск, сервер получает запросы для обработки этих действий и предоставления информации. Это то, что мы в быту называем словом “трафик”.

Мониторинг трафика важен по нескольким причинам:
– ранняя диагностика проблем
– контроль за использованием ресурсов
– управление производительностью
– потребности в локальном масштабировании
– планирование будущего роста
– контроль затрат
Это не полный список причин.

Трафик может количественно меняться двумя способами: резко и плавно. Также трафик зависит от дня недели, времени года, времени суток, событий в мире и тп. Нет такого единственного абсолютного значения, отклонение от которого нужно считать проблемой, всегда есть диапазоны и колебания.

Для таких случаев используется комплексный мониторинг: по абсолютам и по тренду. У них разный принцип работы.

Мониторинг по абсолютному значению

Нужен мониторинг абсолютного значения сверху, а также снизу. Это может быть очень широкий диапазон, потому что верх определяют по максимуму в дни высокого трафика, а низ – по минимуму в дни низкого трафика. Необходимо изучить свою динамику трафика за достаточно большой период и на основании неё выбрать абсолютные значения.

Важно понимать, что если случится внезапный рост трафика в дни с обычно низким трафиком, то из-за ширины диапазона этого можно не увидеть, ведь значения останутся в зоне нормы. Для этого нужен другой мониторинг.

Мониторинг по тренду

Этот вид мониторинга предполагает некоторое накопление данных, и есть множество алгоритмов его работы: сравнение текущего уровня трафика с типичным уровнем для этого дня недели, накопление пятиминутных значений и сравнение их между собой.. здесь каждый сам выбирает нужный ему алгоритм, исходя из доступных технических средств и ситуации.

Важно понимать: если трафик растет достаточно медленно (в течение суток или недели, например), то мониторинг по тренду может не сработать – рост будет слишком плавный, также как плавным может быть и падение трафика. Здесь как раз поможет мониторинг по абсолютным значениям, который может сработать не так быстро, но лучше когда-то, чем никогда.

К сожалению, обычно мониторят только рост трафика, потому что боятся за нагрузку и работоспособность системы, но падение трафика не менее важно: нет трафика = нет пользователей. Нет пользователей = нет денег. Спад трафика может указывать на проблему с доступом пользователей, такую как проблемы с DNS, истек срок действия SSL-сертификатов или неработающая функциональность интерфейса, которая не позволяет пользователям выполнять запросы и решать свои задачи, выпуск новой версии и ещё целая куча разных причин. Но обычно падение трафика не говорит вообще ни о чём хорошем.

Поэтому рецепт такой: трендовый мониторинг делать для нахождения отклонений в типичном поведении, а пороговый мониторинг для крайних случаев, когда трендовый не способен определить отклонения. Мониторить не только рост трафика, но и падение.

11. Мониторинг среднего и min/max

В системах, где много серверов / узлов / нод (выберите любую единицу своей системы), невозможно мониторить каждую единицу. Поэтому для мониторинга значений делают агрегаты: перцентили, медианы и тп. То есть, некоторое среднее по больнице. Это разумный подход, но есть нюанс: обычно есть единичные отклонения, которые в агрегате будут не заметны.

Проблема может быть на одном хосте из сотни, но вы не узнаете об этом. «Но это же всего один хост» скажут многие. Какая разница – он может быть не «одним», а «первым» в череде выхода из строя целой группы. Это совершенно разные ситуации.

Для этого случая полезно иметь мониторинг хотя бы на минимальное значение и на максимальное значение, либо использовать 95ю, 99ю перцентиль и другие виды перцентилей.

Например, если вы мониторите среднее время ответа и используете его для управления масштабированием, то имейте ввиду: половина запросов будет работь дольше этого среднего. Тут возникает очень важный вопрос: а насколько дольше?

В общем, вот что нужно сделать для улучшения ситуации:
– разобраться с перцентилями: что это такое, о чем они говорят
– проанализировать различные значения перцентилей в своей системе
– решить, какую информацию вы хотите получать о системе
– выбрать правильные значения перцентилей для мониторинга

12. Не сажайте слона и моську в одну базу

Мир IT продолжает развиваться быстро. Более того, эта скорость набирает обороты. Чем быстрее вы покажете свою идею в виде продукта, тем больше шансов выиграть в гонке. Здесь менеджеры продукта в прямом смысле соревнуются с инженерами: кто же выиграет – скорость запуска или архитектура?

При появлении очередного проекта, который надо сделать быстро-быстро, первое приходящее в голову звучит так: "Хмм, у нас уже есть в продакшене монга-постгря-мускуль-что-угодно, подселим новую базу для проекта туда!"

Не стоит так делать. А если делаете, то осознавайте последствия и риски. Получается связанность критических компонентов двух совершенно разных проектов. Пойдёте базу обновлять – накосячите и сломаете оба проекта. Придёт большой трафик обновления в один проект, база начнёт тормозить и сломает второй проект.

Вот ещё несколько очевидных проблем проблемы, связанных с такой практикой:

Когда несколько проектов обращаются к одной и той же базе данных, существует повышенный риск нарушения безопасности, так как все данные в базе данных, включая данные других проектов, могут быть скомпрометированы, если в одном проекте будут найдены уязвимости.

По мере роста числа проектов, использующих одну и ту же базу данных, становится сложнее масштабировать её для удовлетворения всех потребностей, так как сложно оптимизировать единую базу данных для противоречивых сценариев.

Управлять контролем доступа и разрешениями для нескольких проектов в рамках одной базы данных сложно, что создаёт новую головную боль при эксплуатации.

Возьмите себе за правило: один проект – одна база данных.

Деньги: С первого взгляда может казаться, что это лишняя работа и лишний расход средств. Помните, пытаясь сэкономить на таких решениях, вы берете эти деньги в кредит у себя будущего под высокий процент.

13. Расселяйте критичные сервисы и непредсказуемые сервисы

Представьте себе ситуацию, что у вас есть один бекенд. Например, он отвечает за оформление заказа на сайте, что является критической функциональностью вашего бизнеса. Нет заказов = нет денег. В какой-то момент времени вы приходите к отличной идее, что хочется получать немного больше информации о том, что делает клиент на сайте. Вы добавляете отправку событий со стороны клиента в свой бекенд. Это ведь такое красивое решение: запрос с клиента проходит через бекенд, обогащается там дополнительной информацией и записывается в специальную базу данных для сбора исторических данных. У вас уже есть опыт и вы учли предыдущий совет про раздельные базы данных.

Данные приходят, вы сделали на этих данных очень красивые дашборды о поведении пользователя… Но в один день что-то идёт не так и весь бекенд ломается от нагрузки, с которой вы ничего не можете сделать. Оказалось, что в последнем утреннем релизе фронтенда закралась очень маленькая ошибочка, в результате которой все загружаемые пользователями страницы начали отправлять десятикратное количество своих событий. И самое печальное, что они продолжают это делать, даже если пользователь не производит никаких действий. В прямом смысле вы сами себе сделали Ddos-атаку.

Эти критически важные сервисы – источник жизненной силы организации, и они должны работать непрерывно для обеспечения бизнеса. Сервисы с неконтролируемой нагрузкой требуют особого внимания для поддержания стабильности и производительности.

Отсюда следует правило: не смешивайте сервисы.

Дополнительные преимущества такого подхода:

Выделенные ресурсы для критически важных сервисов позволяют точно настроить их производительность, обеспечивая максимальную эффективность.

Для раздельных сервисов проще обеспечивать масштабирование.

Разделение сервисов повышает безопасность, ограничивая поверхность атаки и снижая риск для критически важных частей.

Техническое обслуживание и модернизация проводятся с минимальным воздействием на другие сервисы, что снижает количество простоев и сбоев.

Выделенные сервисы облегчают мониторинг и выявление проблем.

Деньги: этот подход позволяет гибко управлять затратами на обеспечение функционирования. Для критических сервисов разумно использовать динамическое выделение ресурсов и резервирование (если это допускает ваша архитектура). Для некритических сервисов это совершенно точно не нужно.

14. Exponential backoff

Ретраи (перезапросы) это такая сущность, которая способна сгладить шероховатости от целого ряда проблем, но при этом таит внутри себя шипы, которые при любом удобном случае добивают жертву, быстро уменьшая её страдания и любые попытки выжить.

Если у вас есть какой-то сервис, в который вы постоянно ходите с ретраями, пытаясь получить ответ – не надо его добивать, когда он уже сломался. Сломанный сервис вполне ясно говорит, что он не может обработать ваш запрос, и возвращает какую-то ошибку. Например, 500 или 503, или что-то еще начинающееся с цифры 5.

Это может быть в нескольких случаях:
– отвалился конкретно этот запрос по "какой-то причине"
– отвалился конкретно этот хост
– отвалился сервис целиком
– и еще масса других вариантов

В каких случаях ретрай сделает хорошо? Только в двух – отвалился конкретный хост с приложением, отвалилась сеть между вами и хостом. В других случаях вы будете ретраями прикладывать сервис больше и больше. Используйте exponential backoff (экспоненциальное откладывание) или любую другую методику, увеличивающую интервал между перезапросами. Таким образом, вы сначала потыкаетесь в несчастную жертву, но со временем дадите ей шанс восстать как феникс и удовлетворить ваши потребности.

С особенным вниманием этот совет стоит изучить разработчикам клиентских приложений… Что может быть лучше, чем ситуация, когда бекенд прилёг, а все клиенты начинают без перерыва пытаться всунуть ему всё новые и новые запросы? Самая интрига в том, что вы ничего не можете с ними сделать в моменте.

15. Учитесь деградировать заранее

Теперь немного про деградацию. Представьте себе, что ваши коллеги-маркетологи запустили рекламную акцию! С кем не бывает… Реклама, дающая новые заказы, это просто чудесно. Коллеги – классные ребята и всегда согласовывают с вами акции, к которым вы заранее готовитесь. Но что-то пошло не так и акция запустилась на сутки раньше, чем вы планировали добавить ресурсов в свою систему. Бекенд быстро сломался, в том числе из-за пользователей, беспрерывно нажимающих “обновить” в браузере. Деньги потрачены зря, вечеринку по поводу успешных продаж придётся отменить.

Совсем не сесть в калошу помогут средства деградации, которые надо сделать заранее. Если заранее не сделали, то сделайте после первого такого инцидента, когда вы задумались, что было бы неплохо иметь запасной парашют.

В случае прихода внезапного трафика у вас не будет никаких вариантов кроме: масштабировать и деградировать. Третий вариант – смириться и переждать – не рассматриваем.

Масштабировать сервис бывает довольно сложно. Сколько нужно будет времени, чтобы развернуть ещё ресурсов – кажется, это порядок минут в лучшем случае (и то, если вы заранее всё предусмотрели). Также важно оценить ваши возможности по оплате этих дополнительных ресурсов.

Деградацию предусмотреть достаточно легко. Это может быть автоматика или ручное управление. Автоматика работает быстро, но в ней могут быть ошибки случайного включения. Ручное управление – медленнее, но с вашим интеллектом и системой принятия решений.

Итак, к вам нагрянули пользователи. Причина вам неизвестна. Сколько это будет длиться – неизвестно. Нужно делать сразу всё: деградировать и масштабировать одновременно!

Деградация – это не "отвечаю через раз", это "отвечаю всегда, но не полной функциональность".

Желательно иметь несколько уровней деградации:

– всё плоховатенько – будем отключать эти малозаметные блоки, типа “сопутствующие товары” и “такие же как вы покупают”

– всё становится хуже – будем отключать функциональность по нарастающей важности, например “дата доставки” или “превью при наведении”

– все совсем плохо – отдаём статику типа “вот наши лучшие товары, на сайте технические работы”

Согласуйте со своим продакт-оунером сценарий деградации, напишите немного кода и сделайте выключатели, которые будете активировать по заранее согласованной с продакт-оунером последовательности. Он тоже хочет, чтобы сайт работал, пусть не в полную силу.

Деньги: заранее оцените ваши возможности по быстрому масштабированию сервиса в случае необходимости. Любое железо стоит деньги. Согласуйте это количество денег с тем, кто отвечает за финансы. А-то может легко получиться так, что вы очень элегантно обработаете рекламную акцию, получите заказы, но все заработанные деньги уйдут на оплату облачных квот.

16. Кэши и заглушки

Это продолжение темы про деградацию. Если у вас есть всякие реал-тайм данные, которые вы готовите на каждый запрос пользователя, имейте в виду: тот сервис, откуда вы их реал-тайм берете, обязательно перестанет работать. Если хотите жить красиво – имейте какой-то универсальный дефолтный вариант ответа. Если вы для каждого пользователя показываете "Такие же как вы покупали вот это", то в случае поломки начните показывать "Топ-5 покупок в нашем магазине". Это точно лучше, чем не показывать ничего.

Этот же способ можно использовать для режима осознанной деградации: если сервису тяжеловато, то переключитесь на показ данных из кешей в тех местах, где он уместен.

Состав этих кешей и заглушек необходимо согласовать с продакт-оунером.

17. Fallback или "последний шанс"

Теперь представим себе ситуацию, что вы уже отмасштабировали всё, что могли, выключили всю функциональность, которую в принципе можно было выключить, и это всё равно не помогает. Здесь появляется Тыква!

Тыква – это народное название для страницы последней надежды (fallback). В беспощадной борьбе с "502: Bad Gateway" (или другими ошибками про невозможность обработать запрос) выигрывает тот, кто сдаётся последним.

Подготовьте какую-то статическую страницу или набор страниц на тот случай, если всё пойдёт совсем плохо, и отправляйте туда пользователей, запросы которых не удалось обслужить. Состав страницы согласуйте с продакт-оунером.

Из чего можно сделать тыкву:

– если это сайт новостей, то периодически генерируйте статическую страницу с "Топ лучших новостей планеты всей"

– если это интернет-магазин, пусть там будет одна страница с вашим лучшим товаром и что-то полезное на javascript

– сделайте на странице "тетрис", в конце концов – это будет хотя бы забавно

– укажите на странице самые полезные данные для посетителя, которые помогут ему решить свою задачу, например: адрес шоурума, номер телефона для записи к специалисту, кнопка для связи через мессенджер…

Важно! В реальности всегда существуют запросы, которые не удалось обслужить ни одним способом. Об этом важно знать, поэтому на количество показов тыквы нужно сделать мониторинг. Когда вы начнете видеть данные о показах тыквы, вы можете с удивлением обнаружить, что в систему приходят запросы, в принципе неспособные корректно работать, например, из-за багов в системе. Удачной охоты!

18. Прогнозируйте нагрузку на смежников

К этому моменту вы уже неплохо подготовились к серьёзным вещам! Хорошо, если ваши коллеги, сервисы которых вы также используете для обработки запросов, иногда встречаются с вами в коридорах, и есть пара минут для обмена новостями про грядущие запуски новых фичей.

Архитектура вашего проекта может быть какой угодно совершенной, но сценарий работы вашей фичи может вполне затрагивать и других.

Например, вы реализовали рассылку уведомлений на мобильный телефон, в тексте которого есть картинка, загружающаяся из хранилища. Конкретно ваша часть работает отлично и даже текст без опечаток. Вы попробовали на паре сотен получателей – всё понравилось, начинаем рассылать на всех, кто есть в нашей базе. Чем быстрее, тем лучше! Это очень круто – отправить миллион уведомлений за несколько минут, а потом рассказывать об этом на конференциях, не правда ли?

Тут нужно учитывать, что при получении уведомления это устройство пойдет за картинкой – это может быть сюрпризом для хранилища этих картинок (предупредите их). Но уведомление не бывает просто с картинкой, оно обычно откручивает статистику показов (предупредите их тоже). Дальше завертелся ураган событий: пользователь увидел уведомление и нажал, чтобы пойти в приложение. Приложение пошло в бэкенд за новыми данными, бэкенд пошёл куда-то ещё, чтобы персонализировать выдачу (пока остановимся на этом), приложение пошло куда-то ещё, чтобы получить настройки пользователя. Скорость вашей рассылки сыграет здесь злую шутку – чем быстрее ваш очень производительный сервис всё отправит, тем больше одновременной нагрузки создадут мобильные клиенты на всю вашу систему.

Осознайте, что будет происходить в таких сценариях, сколько нагрузки на кого будет создано, и поговорите с ними, чтобы они об этом знали и подготовились.

19. Прогнозируйте неответы реалтаймовых источников

Итак, мы подготовились к нашей рассылке, помогли подготовиться коллегам, чьи сервисы будем использовать в процессе обработки запросов. Выпускаем кракена!

Вы завариваете себе ромашковый чай и начинаете удовлетворённо наблюдать за скоростью рассылки, за графиком растущего на ваш сервис трафика, несущего вам богатство… всё отлично!

Система-то сложная – бэкенд ходит реалтайм в другие бэкенды (назовём их источники). В этот момент становится очень важно уметь различать ситуацию разового неответа источника на запрос от ситуации его полной поломки.

В случае разового неответа повтор запроса (перезапрос) в источник даст нужный результат.

А в случае полной поломки перезапрос может сделать ещё хуже. Например, если у него есть очередь, в которую ваш перезапрос будет заботливо сложен в ожидании обработки. Источник лежит – очередь растёт…

Истинные джентльмены собирают статистику неответов и на основании неё считают вероятность успеха запроса: насколько хорошо отвечает источник в принципе за последний интервал времени. Назовём это "вероятность ответа". На основе этих данных вы можете предположить вероятность его ответа на текущий запрос. Если она мала, то, возможно, не стоит и пытаться.

В этом смысле можно варьировать схемы, но суть не меняется – копите и анализируйте данные.

20. Правильно экспериментируйте с сетью

Современные приложения используют такую базовую сущность, как сеть передачи данных. Если вы, конечно, не делаете какое-то особое приложение, у которого требуется обеспечить сетевую изоляцию – в этом случае вам понадобятся другие советы.

Каналы передачи данных имеют сразу несколько свойств, влияющих на работу вашей системы. Например, скорость передачи данных. В конце концов, она может работать, а может не работать. Частота отказов является таким же свойством вашей сетевой инфраструктуры.

Если вы уже начали проводить испытания стабильности вашей системы, то скорее всего у вас уже есть нагрузочное тестирование, тестирование отключения каких-то компонентов. Следующий важный шаг – это изучение влияние работоспособности сети на работу вашей системы.

Интересно, что разные ухудшения в сетевой инфраструктуре способны вызывать разные эффекты в вашей сложной системе. Например, при внезапном исчезновении сети во время обработки запроса один компонент может послать другому компоненту повторную попытку (перезапрос). Но в ситуации, когда один компонент начал получать данные на свой запрос, а сеть в это время отключилась, эффекты могут быть другие. Или представим себе ситуацию, когда один компонент данных начал передавать данные другому компоненту, который их запросил, но делает это неприемлемо медленно – что будет происходить в вашей системе?

Чтобы правильно оценить влияние неполадок в сетевой инфраструктуре на вашу систему, нужно проводить испытания разными способами: отключение части сети, замедление скорости передачи данных, “моргание” сети…

Если вы думаете, что облачная инфраструктура полностью ограждает вас от этих проблем, вы ошибаетесь. Она тоже ломается. Проведите испытания своей системы в локальной среде, на работу которой вы можете повлиять. С какой-то вероятностью вам удастся найти места для улучшения, которые уберегут вашу систему от поломки в реальных условиях.

21. Никому не верьте

Совет для тех, кто уже познал сущность бытия. Какие бы профессионалы вас не окружали, какую бы документацию вы не читали, в какой бы код вы не смотрели, важно знать главное правило – ошибаются все. Код пишут люди, документацию пишут люди, на ваши вопросы тоже отвечает люди. Человек может с полной уверенностью в себе давать ответ на вашу вопрос, даже не подозревая, что он ошибается или что его информация устарела, а может быть он просто стесняется признаться, что он не знает ответа.

Если вы несёте ответственность за работу действительно важной системы, возьмите себе за правило проверять всю новую информацию, которая к вам поступает, и которая может оказать влияние на надёжность вашей системы. В вопросах надёжности нет места таким явлением, как “мне кажется”, “мне так сказали”, “ну скорее всего это так”… Важны только факты.

И даже то, что я тут пишу, тоже проверяйте.

22. Стандартизируйте процессы

Если есть какой-то рискованный процесс, производимый вручную, лучше если он всегда будет проходить по одинаковой схеме, последствия и риски которой вам известны.

Представим себе, что у вас есть многошаговая процедура по перезагрузке нескольких компонентов в системе. По каким-то личным причинам эту процедуру вы проводите вручную. Запишите себе эту конкретную последовательность действий по выполнению процедуры. Не нужно каждый раз пытаться оптимизировать процесс, меняя последовательность, пропуская шаги и добавляя новую энтропию в систему.

Пытаться в каждом случае придумать новый порядок действий, чтобы минимально затронуть ваш хрупкий продакшен, сделать это быстрее, вы рискуете сломать вообще всё, потому что каждый раз сценарий будет новым, с неизвестным вам поведением. Пусть это будет отработанный сценарий, которым вы хорошо владеете и который с высокой вероятностью даст необходимый результат.

23. Инструкции для «Людочки»

Инструкция для аварийных ситуаций и регламентных работ должна быть такая, чтобы не задействовать мозг человека вообще. Как говорил когда-то один из моих боссов: "Я позвоню нашей Люде и она по твоей инструкции должна суметь это сделать". Люда работала у нас на ресепшене. (Сложно сказать, откуда у неё возьмется доступ в продакшен, но идея отражена верно).

Со временем “я позвоню Люде” заменилось на “представь, что ты на шашлыках в лесу”.

Плохими я называю инструкции, которые:
– допускают неоднозначность трактовок
– содержат слова "если"
– содержат в принципе много слов
– изобилуют терминами
– содержат намёки, типа, это и так очевидно
– … ещё много прочих пунктов

Идеальная инструкция – это набор команд, которые надо скопировать и куда-то вставить, и ссылок, по которым нужно перейти, чтобы открылась конкретная страница.

Кстати, если ваша инструкция достигла этой стадии, пора задуматься о механизации.

24. Не пытайтесь самостоятельно защититься от DDoS и готовьтесь к нему заранее

DDoS рано или поздно приходит ко всему, что заслуживает внимания. Лучше думать про него несколько заранее. Если вы планируете защищаться от DDoS с помощью файлика заблокированных ip – вы не планируете от него защищаться. Если у вас какой-то рейтлимитер, который по User-Agent будет блокировать запросы – вы тоже не планируете защищаться. Это будет работать только в случаях школьных экспериментов, хотя тоже может пригодиться.

Признайте себе, что у вас нет экспертизы, чтобы делать это хорошо, иначе этот вид атак себя давно бы уже исчерпал. DDoS лучше научиться переживать, запасая себе способы деградации, кэши, статику – что угодно, чтобы не позориться перед пользователями в час X, а также изучить возможности провайдера или облака, аутсорс-решения, которые "по кнопке" начнут фильтровать ваш входящий трафик. Но этот вопрос нужно изучить заранее, чтобы не бегать потом перед Новым Годом в жару продаж, когда ваш чОрный конкурент решил потратить сотню баксов и вывести ваш интернет-магазинчик из строя на пару часов. Так или иначе, если потеря денег в случае отказа превышает затраты на подготовку и применение, то этим стоит заняться. Слово "заранее" было использовано здесь четыре раза.

25. Все обновления базы пробуйте на тестовом стенде

Это кошмар DBA – "Я запустил построение индекса и всё умерло". К чему я веду: если что-то нужно поковырять в базе, то стоит хотя бы восстановить у себя её копию и сделать это сначала на копии. А если копия будет ещё и под синтетической нагрузкой, то ситуация будет более близка к реальности. Возможно, вас ждут сюрпризы, о которых вы пожалели бы в продакшене.

26. Катите фичу отключенной

Выкатывая новую код с фичей в продакшен, делайте фичу по умолчанию отключенной. Если вы её выкатили включённой и что-то пошло не так, то вы вполне можете обрести ряд проблем с возвращением "как было". А если у вас несколько бекендов и релиз происходит дольше, чем почти мгновенно? Пользователь жмет кнопку, получает один результат, а потом жмет ту же кнопку и получает другой результат, потому что релиз не везде выкатился. Некрасиво, но есть эффекты похуже.

Например, вы выкатили фичу, а она работает не так, как надо. Что делать? Откатить релиз, конечно же. Внезапное исчезновение возможности откатиться, собрать релиз, утерять доступ – может случиться что угодно, приводящее к невозможному исправлению ситуации.

Поэтому: катить выключенной. Включать через какие-то рубильники/конфиги, при включении фичи логировать, что она включилась (иначе вы рискуете не узнать, если вдруг включение не сработало). А для отключения логирования сделать ещё один выключатель. Слишком сложно? Если вы хотите уверенности в своих действиях, придётся это сделать.

27. Исследуйте post-mortem’ы

Если у вас в продакшене произошла какая-то таинственная история, которая "не воспроизводится", приложите достаточно усилий, чтобы разобраться в случившемся. Если вы не понимаете, что именно случилось и не можете этого повторить – это самое худшее развитие событий! Вы не контролируете происходящее и не можете знать, где и когда оно взорвёт ваш продакшен.

Запланируйте потратить на это исследование какое-то разумное количество времени. Нет нужды копаться до истины любой ценой – ситуация может никогда не повториться. Хотя бы убедитесь, что вы приложили достаточно усилий для выявления причины.

Запишите факты, события, скриншоты, логи – всю информацию, которую сможете собрать. Если не получится разобраться в этот раз, то эти данные помогут хотя бы установить связь разных проявлений в будущем.

Потратьте время на предотвращение катастрофы, если она уже начала стучаться в вашу дверь.

28. Устраняйте возможность массовых операций

Если у вас есть инструмент манипуляций с вашим продакшеном типа командной строки или набора любимых скриптов, не оставляйте в них легкодоступную возможность выполнить действие сразу на всём продакшене. Если для запуска команды в консоли вы можете опционально указать некую группу серверов, а по умолчанию операция будет выполнена везде – у вас в руке граната. Вы обязательно ошибётесь и в важный момент забудете указать конкретный хост или группу, опечатаетесь в параметре и шарахнете команду на всё. А чтобы такого не происходило – не надо иметь такую возможность вообще либо нужно делать её чуть сложнее (с дополнительным подтвержением, например).

И самый главный совет: после запуска опасных массовых операций не покидайте рабочее место до тех пор, пока они не отработают и вы не убедитесь в полученном результате.

29. Правильно рассчитывайте запас мощности

Вернёмся немного к вопросам запаса мощностей. Любые простаивающие ресурсы это, без сомнений, затраты – вы за них платите, а никакой пользы они не приносят.

При этом какой-то запас мощности держать всё равно придётся. На что опираться при этих расчётах?

Для начала нужно описать свои сценарии: в каких случаях и какой объем нагрузки нужно выдерживать. Затем нужно оценить сценарий и продолжительность процедуры масшабирования. Если у вас масштабирование устроено по принципу “жмём кнопку, платим сколько угодно, через минуту получаем мощности”, то у вас нет этой проблемы.

Если ваше масштабирование устроено иначе, то запас какой-то иметь придётся, хотя бы на выживание во время масштабирования.

Что включить в расчеты:

– Неожиданный пользовательский трафик, который вы хотите выдерживать. Сюда же входят маркетинговые кампании.

– Выход из строя части инфраструктуры. У любого поставщика услуг есть какие-то гарантии, ознакомьтесь с ними.

– Сломанная часть продакшена в результате неудачного релиза. Здесь всё зависит от вашей процедуры релиза.

– Ещё какие-то особые локальные сценарии.

То есть, нужно понимать, как обслуживать высокий уровень трафика при частично потерянной инфраструктуре из-за провайдера и во время релиза. Вы можете сказать: “Если у нас не будет работать часть инфраструктуры или будет высокий трафик, мы не будем катить релиз”. Обязательно произойдет ситуация, когда это будет необходимо.

Нужно ли запасать мощности на случай DDoS? Я считаю, что в этом нет смысла – всё равно не хватит.

Деньги: любой запас стоит денег. Всю схему запаса нужно посчитать и согласовать с бизнесом: какой трафик в каком случае вы должны обслуживать, а какой не должны. Дальше выстраивать схему резервирования с учётом этой информации.

30. Считайте запас критического пути

Снова про запасы.
Обработка запроса пользователя состоит из трёх основных стадий:
– получение запроса по сети
– обработка бэкендом
– выдача ответа по сети

В стадии "обработка бэкендом" может скрываться бездна этапов: походы в базы данных, сотню других микро- и немикро- сервисов, кэши и тп. Возможно, вы уже молодец и сделали бенчмарк, где измеряете производительность компонентов в синтетической среде. Воспроизвести эквивалент продакшн-среды для бенчмарков это дорогостоящая и трудоёмкая задача – этим очень мало кто занимается.

Если вы делаете бенчмарки в синтетической среде с помощью заглушек, то вы ничего не знаете о производительности системы в реальности. Самым медленным или ограниченным звеном во всей цепи может быть что-то, чему вы вообще не придаёте значения.

Опишите свою схему обработки запроса, проанализируйте пропускную способность каждого компонента и стыки между ними, опишите критический путь выполнения запроса и оцените узкие места.

31. Заведите запасной мониторинг

Этот пункт немного параноидальный, но тоже взялся не просто так, а из очередного увлекательного опыта.

Мониторинг – это система наблюдения за жизненно важными показателям вашей системы. Вообще предполагается, что надежность системы мониторинга должна быть выше, чем у вашей системы.

На самом деле, мониторинг это такой же сервис, как и всё остальное. Ваш мониторинг может быть каким угодно интеллектуально восхитительным, предсказывать что угодно, анализировать зависимости и давать рекомендации дня, но какой от него прок, если он сломался и вы вообще не понимаете, что сейчас в продакшене происходит?

Выделите критические показатели вашего продакшена и сделайте резервный мониторинг – но не делайте такой же, какой уже есть, сделайте на каких-то других технологиях и в другой среде.

Если ваш первый мониторинг сломается в результате автоматического обновления, тогда останется хотя бы второй. Также заведите себе в календаре напоминание "проверить и актуализировать резервный мониторинг".

32. Умейте быстро отключить любой компонент

Был у нас как-то случай… В 00:00 на странице должен был начать отображаться один из блоков, содержащий карточки событий. Код был написан так, что в цикле “while (n < m)” подбирал события до тех пор, пока в ленту не наберётся необходимое количество событий. В один момент в кандидатах было в принципе меньше событий, чем должно было набраться в ленту. На такое, конечно же, никто не рассчитывал. Тут несложно догадаться, что было дальше… В 00:00 блок начал показываться, люди заходили на страницу, но не видели ничего, потому что обслуживающий бекенд уходил в бесконечный цикл. Балансер делал три попытки получить ответ от бекенда; таким образом, на один запрос пользователя три бекенда уходили в бесконечный цикл и уже не могли обслуживать другие запросы. Печаль. В итоге, это приводило к тому, что за небольшое время вообще все бекенды оказывались в бесконечном цикле и по сути ничего не работало. Понятное дело, что в этом случае нужно собирать новую версию и выкатывать её, но это вообще дело небыстрое, когда речь идёт о крупных системах.

Или вот другая история: один из источников данных начал отдавать сломанный json, на котором ломался парсер и всё падало.

Или вот ещё история: один из источников по ошибке начал отдавать такой объём данных в ответе, что в цепочке передачи данных переполнялся один из буферов и снова всё падало.

И вот ещё пример: в одном из блоков был оптимизирован алгоритм сортировки элементов с учетом весов, но когда все веса элементов оказались равными нулю, он начал уходить в бесконечный цикл.

Мораль такова: вообще не знаешь, что и где может произойти, поэтому подготовиться к этому невозможно.

Решение: умейте отключать компоненты в системе. Это намного быстрее, чем собрать и выкатить релиз. Достаточно найти место, которое всё ломает и выключить его, а потом спокойно заниматься решением проблемы, фиксами и релизами.

Для реализации такой штуки есть разные способы:

– админка, через которую вы можете управлять конфигами

– база данных, в которой лежит конфигурация (это удобно, но менее надёжно)

– система управления конфигами, типа Ansible

– файлик в хранилище файлов, который вы туда руками положили, а бекенд его скачивает иногда

– в конце концов, можно файлик закачать прям в контейнер, если нет ничего вообще

Не надо слишком усложнять, выбирайте подходящее для вашей ситуации решение.

33. Ставьте маленькие дефолты

Маленькая заметка о щедрости и здравомыслии.

Представьте себе, что в вашей системе есть какие-то настройки. Или, возможно, вы предоставляете систему другим, и в ней есть какие-то настройки.

Чтобы настройки настраивать, в них нужно писать какие-то значения. Чтобы писать разумные значения, нужно иметь представление о последствиях, что вообще бывает редко. Поэтому в основном люди оставляют значения по умолчанию (дефолты), а когда с ними что-то идёт не так, то идут разбираться, что это за настройки и какие значения там нужно выставлять. Значения по умолчанию для того и созданы – уменьшать когнитивную нагрузку на пользователя.

Возьмите себе за правило: не использовать безлимиты никогда, ставить минимально необходимые значения по умолчанию.

У этого подхода есть несколько позитивных эффектов. Во-первых, люди начинают немного погружаться в смысл настройки, когда им нужно поднять её значение. Начинают хотя бы стремиться к осознанному выбору нового значения. Во-вторых, чем чаще они его поднимают, тем больше задумываются о тенденции.

В-третьих, вы используете эффект привязки – особенность оценки неизвестных числовых значений человеком, из-за которой эта оценка смещается в сторону ранее воспринятых чисел.

Представьте себе, что в вашем продакшене создание контейнеров для новых сервисов происходит через UI, в котором нужно выбрать количество cpu, по умолчанию равное 16 cpu. В итоге у вас все многочисленные контейнеры в продакшене будет размером в 16 cpu, просто потому что никто не будет заниматься их уменьшением в ходе работы и тратить время на выбор более подходящего значения в момент создания, а люди просто будут считать, что контейнер в 16 cpu это вообще нормально и так должно быть.

Представьте себе другую ситуацию: по умолчанию размер контейнера равен 1 cpu. Тогда ваши контейнеры в продакшене будут ровно того размера, который им нужен, потому что люди будут со временем постепенно добавлять в них ресурсов. При этом, контейнер размером в 1 cpu будет считаться нормальным, а контейнеры с 16 cpu будут вызывать вопросы: это же в 16 раз больше, чем значение по умолчанию.

Понятное дело, что у всех разные системы и разные ситуации, но суть к этому моменту уже точно должна быть понятна.