

*Мы посвящаем эту книгу Дэвиду Хассману.
Дэйв был той искрой, которая превратила команду
Chaos Engineering Team в сообщество*

Содержание

Предисловие	12
Введение. Рождение хаос-инжиниринга	15
Часть I. ОБЗОР ПОЛЯ ДЕЯТЕЛЬНОСТИ.....	23
Глава 1. Знакомьтесь: сложные системы.....	24
1.1. Размышления о сложности	24
1.2. Столкновение со сложностью.....	26
1.2.1. Несоответствие между бизнес-логикой и логикой приложения	26
1.2.2. Лавина повторных запросов пользователей.....	29
1.2.3. Замораживание кода на праздники.....	33
1.3. Противодействие сложности.....	36
1.3.1. Случайная сложность	36
1.3.2. Намеренная сложность	37
1.4. Принятие сложности.....	39
Глава 2. Навигация по сложным системам.....	41
2.1. Динамическая модель безопасности.....	41
2.1.1. Экономика	42
2.1.2. Нагрузка.....	42
2.1.3. Безопасность	42
2.2. Экономические факторы сложности	44
2.2.1. Состояния	45
2.2.2. Отношения	45
2.2.3. Окружение	45
2.2.4. Обратимость.....	46
2.2.5. Экономические факторы сложности и программное обеспечение	46
2.3. Системный подход.....	47
Глава 3. Обзор принципов хаос-инжиниринга	49
3.1. Что такое хаос-инжиниринг	49
3.1.1. Эксперименты или тестирование?	50
3.1.2. Функциональный контроль или аттестация?	51
3.2. Чем не является хаос.....	52
3.2.1. Разрушающее тестирование производства.....	52
3.2.2. Антихрупкость.....	53
3.3. Ключевые принципы хаос-инжиниринга	54
3.3.1. Построение гипотезы о стабильном поведении.....	54
3.3.2. Моделирование различных событий реального мира	55
3.3.3. Выполнение экспериментов на производстве	56

3.3.4. Автоматизация непрерывного запуска экспериментов	56
3.3.5. Минимизация радиуса поражения	57
3.4. Будущее «Принципов»	59
Часть II. ПРИНЦИПЫ ХАОСА В ДЕЙСТВИИ	61
Глава 4. Slack и островок спокойствия среди хаоса	63
4.1. Настройка методов хаоса под свои нужды.....	64
4.1.1. Подходы к проектированию старых систем	64
4.1.2. Подходы к проектированию современных систем	65
4.1.3. Предварительная подготовка отказоустойчивости.....	65
4.2. Disasterpiece Theater	66
4.2.1. Цели экспериментов.....	67
4.2.2. Антицели	67
4.3. Процесс проверки по шагам	68
4.3.1. Подготовка эксперимента	68
4.3.2. Эксперимент.....	71
4.3.3. Подведение итогов.....	74
4.4. Как развивался Disasterpiece Theater.....	74
4.5. Как получить одобрение руководства	75
4.6. Результаты	76
4.6.1. Избегайте несогласованности кеша.....	76
4.6.2. Пробуйте и еще раз пробуйте	77
4.6.3. Невозможность как результат	77
4.7. Вывод.....	78
Глава 5. Google DiRT: тестирование аварийного восстановления	79
5.1. Жизненный цикл теста DiRT	81
5.1.1. Правила взаимодействия	82
5.1.2. Что следует проверить	86
5.1.3. Как выполнить тестирование.....	93
5.1.4. Сбор результатов.....	95
5.2. Объем тестов в Google.....	96
5.3. Вывод	99
Глава 6. Вариативность и приоритеты экспериментов в Microsoft	101
6.1. Почему все так сложно?	101
6.1.1. Пример неожиданных осложнений	102
6.1.2. Простая система – лишь вершина айсберга	103
6.2. Категории результатов эксперимента.....	104
6.2.1. Известные события / непредвиденные последствия	105
6.2.2. Неизвестные события / неожиданные последствия	106
6.3. Расстановка приоритетов отказов	107
6.3.1. Исследуйте зависимости	108

6.4. Глубина варьирования	109
6.4.1. Вариативность отказов	109
6.4.2. Объединение вариативности и расстановки приоритетов	111
6.4.3. Расширение вариативности до зависимостей	111
6.5. Развёртывание масштабных экспериментов	112
6.6. Вывод	113
Глава 7. Как LinkedIn заботится о пользователях	115
7.1. Учитесь на примерах катастроф	116
7.2. Детализованные эксперименты	117
7.3. Масштабные, но безопасные эксперименты	119
7.4. На практике: LinkedOut	120
7.4.1. Режимы отказа	121
7.4.2. Использование LiX для нацеливания экспериментов	123
7.4.3. Браузерное расширение для быстрых экспериментов	126
7.4.4. Автоматизированные эксперименты	128
7.5. Вывод	130
Глава 8. Развитие хаос-инжиниринга в Capital One	131
8.1. Практический опыт Capital One	132
8.1.1. Слепое тестирование устойчивости	132
8.1.2. Переход к хаос-инжинирингу	133
8.1.3. Хаос-эксперименты в CI/CD	134
8.2. Чего нужно остерегаться при разработке эксперимента	135
8.3. Инструментарий	136
8.4. Структура команды	137
8.5. Продвижение хаос-инжиниринга	139
8.6. Вывод	139
Часть III. ЧЕЛОВЕЧЕСКИЕ ФАКТОРЫ	141
Глава 9. Формирование предвидения	143
9.1. Хаос-инжиниринг и отказоустойчивость	144
9.2. Этапы рабочего цикла хаос-инжиниринга	144
9.2.1. Разработка эксперимента	145
9.3. Инструменты для разработки хаос-экспериментов	146
9.4. Эффективное внутреннее партнерство	148
9.4.1. Организация рабочих процедур	149
9.4.2. Обсуждение предмета эксперимента	151
9.4.3. Построение гипотезы	152
9.5. Вывод	154
Глава 10. Гуманистический хаос	156
10.1. Люди в системе	156
10.1.1. Значение человека в социотехнических системах	157
10.1.2. Организация – это система систем	158

10.2. Инженерно-адаптивный потенциал	158
10.2.1. Обнаружение слабых сигналов	159
10.2.2. Неудача и успех, две стороны одной монеты.....	160
10.3. Применение принципов хаос-инжиниринга на практике	160
10.3.1. Построение гипотезы	161
10.3.2. Варьирование событий реального мира	161
10.3.3. Минимизация радиуса поражения	162
10.3.4. Пример 1: игровые дни.....	163
10.3.5. Коммуникации и сетевая задержка в организациях	165
10.3.6. Пример 2: связь между точками	166
10.3.7. Лидерство как новое свойство системы	169
10.3.8. Пример 3: изменение базового предположения	170
10.3.9. Безопасная организация хаоса	172
10.3.10. Все, что вам нужно, – это высота и направление	173
10.3.11. Замыкайте петли обратной связи	173
10.3.12. Если вы не ошибаетесь, вы не учитесь	174
Глава 11. Роль человека в системе.....	175
11.1. Эксперименты: почему, как и когда	176
11.1.1. Почему	176
11.1.2. Как	177
11.1.3. Когда.....	178
11.1.4. Распределение функций, или Каждый хорош по-своему	179
11.1.5. Миф замещения	181
11.2. Вывод	183
Глава 12. Проблема выбора эксперимента и ее решение	184
12.1. Выбор экспериментов.....	184
12.1.1. Случайный поиск	186
12.1.2. Настало время экспертов.....	186
12.2. Наблюдаемость системы	191
12.2.1. Наблюдаемость и интуиция	192
12.3. Вывод	194
Часть IV. ФАКТОРЫ БИЗНЕСА	195
Глава 13. Рентабельность хаос-инжиниринга	196
13.1. Краткосрочный эффект хаос-инжиниринга	196
13.2. Модель Киркпатрика	197
13.2.1. Уровень 1: реакция.....	197
13.2.2. Уровень 2: обучение	198
13.2.3. Уровень 3: перенос	198
13.2.4. Уровень 4: результаты.....	199
13.3. Альтернативный вариант оценки рентабельности	199
13.4. Побочная отдача от инвестиций	201
13.5. Вывод	202

Глава 14. Открытые умы, открытая наука и открытый хаос	203
14.1. Совместное мышление	203
14.2. Открытая наука, открытый исходный код	205
14.2.1. Открытые хаос-эксперименты.....	206
14.2.2. Обмен результатами и выводами	208
14.3. Вывод	208
Глава 15. Модель зрелости хаоса	209
15.1. Внедрение	209
15.1.1. От кого исходит идея внедрения.....	210
15.1.2. Какая часть организации участвует в хаос-инжиниринге	211
15.1.3. Обязательные условия	212
15.1.4. Препятствия для внедрения	213
15.1.5. Освоение	214
15.2. Карта состояния хаос-инжиниринга	219
Часть V. ЭВОЛЮЦИЯ	221
Глава 16. Непрерывная проверка	223
16.1. Происхождение непрерывной проверки.....	223
16.2. Разновидности систем непрерывной проверки	225
16.3. CV в реальной жизни: ChAP	227
16.3.1. Выбор экспериментов в ChAP	227
16.3.2. Запуск экспериментов в ChAP	228
16.3.3. ChAP и принципы хаос-инжиниринга	228
16.3.4. ChAP как непрерывная проверка.....	229
16.4. Непрерывная проверка в системах рядом с вами	229
16.4.1. Проверка производительности	230
16.4.2. Артефакты данных	230
16.4.3. Корректность	230
Глава 17. Поговорим о киберфизических системах	232
17.1. Происхождение и развитие киберфизических систем	233
17.2. Слияние функциональной безопасности с хаос-инжинирингом	234
17.2.1. FMEA и хаос-инжиниринг	236
17.3. Программное обеспечение в киберфизических системах	236
17.4. Хаос-инжиниринг как следующий шаг после FMEA	238
17.5. Эффект щупа	241
17.5.1. Решение проблемы щупа	242
17.6. Вывод	244
Глава 18. НОР с точки зрения хаос-инжиниринга	246
18.1. Что такое НОР?	246
18.2. Ключевые принципы НОР	247
18.2.1. Принцип 1: ошибка – это норма	247
18.2.2. Принцип 2: вина ничего не исправляет	247

18.2.3. Принцип 3: контекст определяет поведение	248
18.2.4. Принцип 4: обучение и улучшение имеют жизненно важное значение	249
18.2.5. Принцип 5: важны осмысленные ответы	249
18.3. Хаос-инжиниринг в мире НОР	249
18.3.1. Практический пример хаос-инжиниринга в мире НОР	251
18.4. Вывод	253
Глава 19. Хаос-инжиниринг и базы данных	254
19.1. Зачем нам нужен хаос-инжиниринг?	254
19.1.1. Надежность и стабильность	254
19.1.2. Пример из реального мира	255
19.2. Применение хаос-инжиниринга	257
19.2.1. Наш особый подход к хаос-инжинирингу	257
19.2.2. Внедрение отказов	258
19.2.3. Отказы приложений	258
19.2.4. Ошибки процессора и памяти	259
19.2.5. Отказы сети	259
19.2.6. Внедрение ошибок в файловую систему	260
19.3. Обнаружение сбоев	261
19.4. Автоматизация хаоса	262
19.4.1. Автоматизированная платформа для экспериментов Schrodinger	262
19.4.2. Рабочий процесс на платформе Schrodinger	264
19.5. Вывод	264
Глава 20. Хаос-инжиниринг в информационной безопасности	266
20.1. Современный подход к безопасности	267
20.1.1. Человеческий фактор и отказы	267
20.1.2. Устраните легкодоступные цели	269
20.1.3. Петли обратной связи	270
20.2. Хаос-инжиниринг и новая методология безопасности	271
20.2.1. Проблемы с Red Teaming	272
20.2.2. Проблемы с Purple Teaming	272
20.2.3. Преимущества хаос-инжиниринга в кибербезопасности	273
20.3. Игровые дни в кибербезопасности	274
20.4. Пример инструмента безопасности: ChaoSlingr	274
20.4.1. История ChaoSlingr	275
20.5. Вывод	277
Заключение	279
Предметный указатель	282

Предисловие

Революция хаос-инжиниринга свершилась! Тысячи компаний всех форм и размеров, достигшие разного уровня развития, используют хаос-инжиниринг в качестве основного инструмента тестирования и контроля, чтобы сделать свои продукты и услуги более безопасными и надежными. Существует множество ресурсов по этой теме, в частности выступления на конференциях, но ни один из них не дает полной картины.

Нора и Кейси задумали написать самую полную книгу о хаос-инжиниринге. Это весьма непростая задача, учитывая широту и разнообразие отрасли и развивающийся характер дисциплины. В этой книге мы попытаемся охватить историю рождения хаос-инжиниринга, фундаментальные теории, лежащие в его основе, определения и принципы, примеры реализации в масштабных вычислительных системах, примеры за пределами традиционного программного обеспечения и будущее, которое, на наш взгляд, ожидает подобные практики.

СОГЛАШЕНИЕ О ТЕРМИНАХ В РУССКОМ ПЕРЕВОДЕ КНИГИ

В этой книге мы используем уже устоявшееся в среде специалистов кириллическое написание термина *хаос-инжиниринг* для обозначения стратегического подхода к тестированию и идеологии хаоса в целом, а термины Chaos, Chaos Engineering Team, Chaos Monkey и другие слова в исходном английском написании с заглавной буквы обозначают названия рабочих групп, инструментов и технологий, а также зарегистрированные марки компаний и их продуктов.

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И СОГЛАШЕНИЯ, ПРИНЯТЫЕ В КНИГЕ

В книге используются следующие типографские соглашения:

- *курсив* – для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также имен и расширений файлов и каталогов;
- *моноширинный шрифт* – для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите нам о ней главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Благодарности

Мы можем назвать бесчисленное множество людей, которые вложили в подготовку этой книги время и силы, а также оказали эмоциональную поддержку авторам, редакторам и соавторам. Трудно переоценить объем помощи, полученной нами при создании сборника, включающего целых шестнадцать авторов (основные авторы Нора и Кейси, а также четырнадцать соавторов). Мы ценим все усилия соавторов, их терпение по отношению к нам, когда мы дорабатывали идеи и объем глав, а также помочь в процессе редактирования.

Нам повезло работать с замечательными редакторами и персоналом в O'Reilly. Амелия Блевинс (Amelia Blevins), Вирджиния Уилсон (Virginia Wilson), Джон Девинс (John Devins) и Никки Макдональд (Nikki McDonald) сыграли важную роль в создании этой книги. Во многих отношениях эта книга – произведение, созданное Амелией и Вирджинией так же, как и авторами. Спасибо за ваше терпение с нами и за многие, многие переносы сроков.

Мы ценим энтузиазм наших рецензентов: Уилла Гальего (Will Gallego), Райана Франца (Ryan Frantz), Эрика Доббса (Eric Dobbs), Лейна Десборо (Lane Desborough), Рэндала Хансена (Randal Hansen), Майкла Кехо (Michael Kehoe), Матиаса Лафельдта (Mathias Lafeldt), Барри О'Рейли (Barry O'Reilly), Синди Сридхарана (Cindy Sridharan) и Бенджамина Уилмса (Benjamin Wilms). Ваши комментарии, предложения и исправления значительно улучшили качество этой работы. Кроме того, ваши советы привели нас к дополнительным исследованиям, которые мы включили в книгу. По сути, эта книга стала результатом нашей совместной работы с вами.

Мы многим обязаны нашим соавторам, это: Джон Алспоу (John Allspaw), Питер Альваро (Peter Alvaro), Натан Ашбахер (Natan Aschbacher), Джейсон Каухун (Jason Caagoon), Раджи Чокайян (Raji Chockaiyan), Ричард Кроули (Richard Crowley), Боб Эдвардс (Bob Edwards), Энди Флиннер (Andy Fleener), Расс Майлз (Russ Miles), Аарон Ринхарт (Aaron Rinhart), Логан Розен (Logan Rosen), Олег Сурмачев, Лю Тан (Lu Tang) и Хао Вэн (Hao Weng). Очевидно, что данной книги не было бы без вас. Каждый из вас внес необходимый и принципиально важный вклад в содержание. Мы ценим вас как единомышленников и друзей.

Мы хотим поблагодарить Дэвида Хассмана (David Hassman), Кента Бека (Kent Beck) и Джона Алспоу. Дэвид, которому посвящена эта книга, призвал нас пропагандировать идеологию хаос-инжиниринга за пределами нашего ограниченного профессионального сообщества в Кремниевой долине. Во многом благодаря его вовлеченности и поддержке хаос-инжиниринг стал осозаемой «вещью» – самостоятельной дисциплиной в широком мировом сообществе разработчиков программного обеспечения. В свою очередь, Кент Бек призвал нас воспринимать хаос-инжиниринг как инструмент, намного более серьезный, чем мы думали, способный изменить взгляды людей на создание, развертывание и эксплуатацию программного обеспечения. Джон Алспоу дал нам фундаментальные основы хаос-инжиниринга, подтолкнув нас заняться изучением человеческого фактора и систем безопасности в Лундском университете в Швеции. Он познакомил нас с областью Resilience Engineering (технологии отказоустойчивости), которая послужила фундаментом для хаос-инжиниринга и является тем объективом, через который мы рассматриваем проблемы надежности (включая доступность и безопасность), когда изучаем социально-технические системы, такие как широкомасштабное программное обеспечение. Все руководители и коллеги по программе в Лунде глубоко повлияли на наше мышление, особенно Йохан Бергстрём (Johan Bergstrom) и Энтони Смоукер (Antony Smoker).

Мы благодарим всех вас за влияние, которое вы оказали на нас, за то, что вдохнули в нас смелость продвигать идею хаос-инжиниринга в общество, и за то неизгладимое влияние, которое оказали на наше мировоззрение.

Введение. Рождение хаос-инжиниринга

Хаос-инжиниринг все еще остается относительно новым направлением в разработке программного обеспечения. В этом введении мы расскажем историю метода, начиная со скромного первого использования и заканчивая нынешней эпохой, когда все основные представители информационной отрасли переняли идеологию хаос-инжиниринга в той или иной форме. За последние три года вопрос «Должны ли мы заниматься хаос-инжинирингом?» превратился в вопрос «Каким образом лучше всего внедрить хаос-инжиниринг?».

История нашей зарождающейся дисциплины объясняет, как мы перешли от первого вопроса ко второму. Мы хотим не просто перечислить даты и события. Мы хотим рассказать живую историю о том, как это произошло, чтобы вы поняли, почему это произошло именно так, и какие уроки вы можете извлечь из этого пути, чтобы получить максимальную отдачу от практического применения.

История начинается в Netflix, где работали авторы этой книги Кейси Розенталь и Нора Джонс, когда команда Chaos Team создавала и внедряла технологию Chaos Engineering¹. Netflix получил от этой технологии ощутимую выгоду, и как только другие компании увидели это, вокруг новой дисциплины возникло сообщество, которое распространило ее по всему миру.

МЕНЕДЖМЕНТ В ВИДЕ КОДА

Начиная с 2008 года Netflix приступил к масштабному переходу² с собственного центра обработки данных на внешний облачный сервис. В августе того же года из-за крупного сбоя базы данных в центре обработки данных Netflix не мог отправлять DVD-диски в течение трех дней. Это было до того, как потоковое видео стало вездесущим; рассылка DVD по почте составляла основу бизнеса компании.

В то время считалось, что центр обработки данных в силу своей архитектуры содержит несколько глобальных точек отказа, таких как большие

¹ Кейси Розенталь в течение трех лет создавал технологию и руководил командой разработчиков Chaos в Netflix. Нора Джонс присоединилась к команде разработчиков Chaos на раннем этапе в качестве инженера и технического лидера. Она отвечала за важные архитектурные решения в создаваемых инструментах, а также занималась внедрением.

² Yury Izrailevsky, Stevan Vlaovic, Ruslan Meshenberg. Completing the Netflix Cloud Migration // Netflix Media Center, Feb. 11, 2016, <https://oreil.ly/c4YTI>.

базы данных и вертикально масштабируемые компоненты. Переход к облаку означает использование горизонтально масштабируемых компонентов, что уменьшает количество и влияние точек отказа.

Увы, все пошло не по плану. Прежде всего потребовалось целых восемь лет, чтобы избавиться от центра обработки данных. Хотя теоретически облачные технологии больше отвечали интересам компании, переход к горизонтально масштабируемым облачным решениям не привел к желаемому повышению времени безотказной работы потокового сервиса¹.

Чтобы понять причину, мы должны вспомнить, что в 2008 году облачный сервис Amazon Web Services (AWS) был значительно менее зрелым, чем сейчас. Облачные вычисления еще не были товаром, а вариантов развертывания по умолчанию, которые мы имеем сегодня, просто не существовало. В то время облачный сервис действительно имел много особенностей, и одна из этих особенностей заключалась в том, что экземпляры² иногда прекращали свою работу без предупреждения. Такая форма сбоя считалась редкостью в центре обработки данных, где большие мощные машины тщательно обслуживались, а специфические особенности конкретных машин были хорошо известны. В облачной среде, где такую же вычислительную мощность обеспечивали за счет большого количества относительно слабых машин, работающих на обычном «железе», это, к сожалению, было обычным явлением.

Методы построения систем, устойчивых к такой форме отказа, были хорошо известны. Можно перечислить полдюжины распространенных методов, которые помогают системе автоматически компенсировать внезапный выход из строя одного из компонентов: избыточные узлы в кластере, ограничение области поражения путем увеличения количества узлов и уменьшения относительной мощности каждого узла, избыточное развертывание в разных регионах, автоматическое масштабирование и автоматизация обнаружения неисправностей и т. д. Конкретные средства, способные сделать систему нечувствительной к выходу экземпляров из строя, не имели значения. Они могли даже быть разными в зависимости от контекста системы. Важно было это сделать как можно скорее, потому что потоковый сервис столкнулся с ограничением доступности из-за высокой частоты отказов облачных экземпляров. В некотором смысле Netflix просто раздробил и умножил эффект единой точки отказа.

Однако Netflix отличался от других разработчиков масштабного программного обеспечения. Он активно продвигал культурные принципы, которые вытекают из уникальной философии управления компании и сильно повлияли на подходы к решению проблемы надежности, например:

- Netflix нанимал только старших разработчиков, которые имели опыт работы в той роли, для которой их наняли;
- Netflix предоставил всем разработчикам полную свободу действий, необходимых для решения проблемы, вкупе с ответственностью за любые последствия, связанные с этими действиями;

¹ В этой книге мы будем считать критерием доступности системы «время безотказной работы», или «аптайм» (uptime).

² В облачной технологии термин «экземпляр» соответствует виртуальной машине или серверу в предшествующем отраслевом языке.

- важно отметить, что Netflix полностью делегировал людям, делающим свою работу, право принятия решений в рамках этой работы;
- руководство не указывало отдельным исполнителям, что делать, а вместо этого позаботилось о том, чтобы исполнители поняли проблемы, которые необходимо решить. Затем исполнители рассказали руководству, как они планируют решить эти проблемы, и вместе приступили к работе над решением;
- по-настоящему продуктивные команды хорошо сбалансированы и мало связаны друг с другом. Если у всех одинаковое понимание цели, меньше усилий расходуется на процессы, формальное согласование или управление задачами.

Эти принципы являются частью производственной культуры Netflix и оказали любопытное влияние на зарождение идеи хаос-инжиниринга. Поскольку в задачу руководства не входили прямые указания исполнителям, в Netflix не нашлось какого-то одного человека, команды или группы, которые диктовали бы остальным разработчикам, как писать свой код. Хотя разработка полдюжины общих шаблонов для написания сервисов, устойчивых к выпадению экземпляров, – вполне решаемая задача, культура компании не позволяла приказать всем разработчикам строго следовать этим инструкциям.

Netflix пришлось искать другой путь.

Рождение Chaos Monkey

Разработчики перепробовали много разных подходов, но сработало как надо и осталось жить только одно решение – Chaos Monkey. Это очень простое приложение просматривает список кластеров, выбирает один случайный экземпляр из каждого кластера и внезапно отключает его в течение рабочего дня. И это происходит каждый рабочий день.

Звучит жестоко, но никто не ставил перед собой цель разрушить коллег. Разработчики знали, что этот тип отказа – выпадающие экземпляры – в любом случае произойдет с каждым кластером. Приложение Chaos Monkey дало им возможность проактивно проверить устойчивость каждого кластера и делать это в рабочее время, чтобы сотрудники могли реагировать на любые потенциальные последствия, когда у них есть для этого ресурсы, а не в 3 часа ночи, когда на телефонах обычно выключен звук. Увеличение частоты до одного раза в день действует наподобие регрессионного теста, гарантирующего, что инженеры не столкнутся с внезапным обвалом системы по причине такого отказа в процессе повседневных обновлений.

Опыт Netflix говорит, что это решение не сразу стало популярным. Был короткий период времени, когда инженеры по эксплуатации недовольно ворчали про Chaos Monkey. Но, похоже, идея сработала, и все больше и больше команд в конечном итоге стали брать метод на вооружение.

Вся суть этого простого приложения заключается в том, что оно взяло на себя ответственность за создание проблемы – исчезающие экземпляры влияют на доступность сервисов – и сделало это головной болью для каждого раз-

работчика. Как только проблема встала в полный рост, разработчики сделали то, что они обычно делают лучше всего: они решили проблему.

Действительно, когда Chaos Monkey каждый день неумолимо (хотя и управляемо) ронял какую-то службу, разработчики не могли спокойно работать, пока не решили эту проблему. И не важно, как они это сделали. Возможно, они добавили избыточность, может быть, автоматизировали масштабирование или пересмотрели шаблоны архитектуры. Это не имеет значения. На самом деле важно лишь то, что проблема была решена быстро и с немедленно ощутимыми результатами.

Достигнутый результат вырос из ключевого принципа культуры Netflix «высокая согласованность, слабая зависимость». Вредоносная обезьяна хаоса заставила всех быть в высшей степени ориентированными на достижение цели, максимально согласованно работать над решением проблемы, но не связывать себе руки в выборе решений.

Chaos Monkey – это прием менеджмента, реализованный в виде кода. Идея, стоящая за ним, выглядела необычно и слегка подозрительно, поэтому Netflix завел на эту тему отдельный блог. Довольно быстро Chaos Monkey стал популярным проектом с открытым исходным кодом и даже инструментом подбора персонала, благодаря которому Netflix предстал на рынке труда в новой роли – как идеолога творческой культуры разработки, а не просто развлекательной компании. Короче говоря, приложение Chaos Monkey стало признанным успехом и примером творческого подхода к риску как части культурной самобытности Netflix.

А теперь перемотаем время вперед до 24 декабря 2012 года, в канун Рождества¹. В этот день в облаке AWS произошел перекос эластичных балансировщиков нагрузки (elastic load balancer, ELB). Эти компоненты анализируют поступающие запросы и распределяют трафик по вычислительным экземплярам, на которых развернуты службы. Когда балансировщики вышли из строя, прекратилась обработка новых запросов. Поскольку плоскость управления Netflix работала на AWS, клиенты не могли выбирать видео и запускать потоковую передачу.

Беда приключилась в наихудшее время. В канун Рождества Netflix собирался занять центральное место за столом, ведь первые пользователи хотели показать своей семье и гостям, как легко смотреть настоящие фильмы через интернет. Вместо этого бедные члены семьи и родственники были вынуждены разговаривать друг с другом, не отвлекаясь на контентную библиотеку Netflix.

Netflix испытал глубокий болевой шок. Авария стала ударом по имиджу компании и чувству гордости за технологии. Вдобавок сотрудникам компании совсем не понравилось, когда их вытащили из-за рождественского стола, чтобы наблюдать, как AWS спотыкается и снова падает в процессе восстановления.

Приложение Chaos Monkey помогло успешно решить проблему выпадающих экземпляров. Однако это локальная проблема. Можно ли построить

¹ Adrian Cockcroft. A Closer Look at the Christmas Eve Outage // The Netflix Tech Blog, Dec. 31, 2012, <https://oreil.ly/wCftX>.

нечто подобное для решения проблемы выпадающих регионов? Будет ли это работать в очень, очень большом масштабе?

НАСТАЛО ВРЕМЯ РОСТА

Каждое взаимодействие устройства клиента с потоковой службой Netflix осуществляется через *плоскость управления*. Это функциональность, развернутая в AWS. Как только выбрано потоковое видео, данные для него начинают поступать из частной сети Netflix, которая на сегодняшний день является самой большой сетью доставки контента в мире.

Перебои в канун Рождества вновь привлекли внимание руководства и специалистов компании к поиску активного подхода по обслуживанию трафика для плоскости управления. Теоретически трафик для клиентов в Западном полушарии можно разделить между двумя регионами AWS, по одному на каждом побережье. Если в одном из регионов происходит сбой, необходимо оперативно смасштабировать инфраструктуру другого региона и перебросить туда все запросы.

Это решение коснулось каждого аспекта потокового сервиса. Существует задержка распространения данных между побережьями. Некоторым службам пришлось доработать свои технологии, чтобы обеспечить согласованность между побережьями, придумать новые стратегии совместного использования состояний и т. д. Конечно, это сложная техническая задача.

И опять же, в структуре Netflix нет органа разработки, выдающего всем программистам какое-то готовое централизованное решение, которое гарантированно справилось бы с региональными сбоями. Вместо этого команда хаос-инженеров, заручившись поддержкой высшего руководства, координировала усилия различных команд.

Чтобы убедиться, что все эти команды готовы к решению задачи, была создана процедура перевода региона в автономный режим. Разумеется, провайдер AWS не позволил бы Netflix перевести целый регион в автономный режим из-за наличия других клиентов в регионе, поэтому аварию пришлось моделировать программными средствами. Процедура получила название Chaos Kong.

Первые несколько запусков процедуры Chaos Kong были весьма увлекательным приключением, когда специалисты собирались в «боевом штабе» и следили за всеми показателями потокового сервиса, а восстановление продолжалось часами. Затем запуски Chaos Kong пришлось прекратить на несколько месяцев, так и не добившись полного переноса трафика в один регион, потому что удалось выявить множество проблем, которые передали владельцам сервисов для устранения. В конце концов работа по переброске трафика была полностью отлажена и формализована и превратилась в обычную работу команды инженеров по управлению трафиком. Тем не менее Chaos Kong продолжали регулярно запускать, чтобы убедиться, что у Netflix есть актуальный план действий на случай, если какой-либо регион рухнет.

И действительно, потом было много случаев, когда из-за проблем со стороны Netflix или из-за неполадок в AWS какой-то один регион страдал от значительных простоев. В этих случаях срабатывал протокол обработки отказа региона, используемый в Chaos Kong. Выгода от инвестиций в новую технологию была очевидна¹.

К сожалению, переброска трафика при отказе региона в лучшем случае занимала около 50 минут из-за сложности интерпретации данных и необходимости ручного вмешательства. Несколько увеличив частоту запусков Chaos Kong, что, в свою очередь, заставило поднять инженерные стандарты в отношении отказоустойчивости регионов, команда инженеров по управлению трафиком смогла отладить новый процесс и в конечном итоге сократила обработку отказа региона до шести минут².

Вот мы и подошли к 2015 году. В распоряжении Netflix на тот момент были Chaos Monkey и Chaos Kong. Первый работал в ограниченном масштабе с выпадающими экземплярами, а второй – в большом масштабе с выпадающими регионами. Оба инструмента появились благодаря технической культуре компании и внесли очевидный вклад в доступность сервиса на этом этапе.

ФОРМАЛИЗАЦИЯ ДИСЦИПЛИНЫ

Брюс Вонг создал команду разработчиков Chaos в Netflix в начале 2015 года и передал задачу по разработке устава и дорожной карты Кейси Розенталю. Не совсем уверенный, во что он ввязался (его первоначально наняли для управления командой по обслуживанию трафика, и какое-то время он параллельно работал в двух командах), Кейси обошел Netflix, спрашивая, что люди думают о хаос-инжиниринге.

Обычно ответом было что-то вроде «хаос-инжиниринг – это когда мы специально ломаем разные места в производстве». Теперь это звучит круто и может стать отличным дополнением к резюме в профиле LinkedIn, но все-таки это пустое определение. Чтобы сломать что-нибудь в производстве, много ума не надо, и это может сделать любой сотрудник Netflix, имеющий доступ к терминалу, только вряд ли это принесет выгоду компании.

Кейси усадил свою команду за стол, чтобы разработать формальное определение хаос-инжиниринга. Они искали ясные ответы на вопросы:

- Как звучит определение хаос-инжиниринга?
- В чем суть хаос-инжиниринга?
- Как я могу убедиться, что действительно занимаюсь именно этим?
- Как я могу улучшить свои навыки?

Примерно через месяц работы над своего рода манифестом они разработали «Принципы хаос-инжиниринга». Дисциплина получила формальную оболочку.

¹ *Ali Basiri, Lorin Hochstein, Abhijit Thosar, and Casey Rosenthal. Chaos Engineering Upgraded // The Netflix Technology Blog, Sept. 25, 2015, <https://oreil.ly/UJ5yM>.*

² *Luke Kosewski et al. Project Nimble: Region Evacuation Reimagined // The Netflix Technology Blog, March 12, 2018, <https://oreil.ly/7bafg>.*

Формальное определение выглядело так: «Хаос-инжиниринг – это дисциплина экспериментов с распределенной системой, призванных подтвердить способность системы противостоять турбулентным условиям рабочей среды». Отсюда следует, что это особая форма экспериментов, которая стоит отдельно от *тестирования*.

Цель хаос-инжиниринга в первую очередь заключается в укреплении доверия к системе. Это полезно помнить, так что если вам не нужна уверенность в системе, то это не для вас. Если у вас есть другие способы укрепления доверия, вы можете взвесить, какой метод наиболее эффективен.

В определении также упоминаются «турбулентные условия рабочей среды», чтобы подчеркнуть, что речь идет не о создании хаоса. Хаос-инжиниринг – это прием, который делает видимым хаос, изначально присущий системе.

Далее «Принципы» описывают базовый шаблон для экспериментов, который в значительной степени соответствует принципу фальсификации Карла Поппера. Поэтому хаос-инжиниринг больше похож на науку, чем на технологию.

И наконец, в «Принципах» перечислены пять ключевых шагов, которые устанавливают золотой стандарт хаос-инжиниринга:

- построить гипотезу о стабильном поведении;
- продумать различные события реального мира;
- провести эксперименты в производстве;
- автоматизировать эксперименты для непрерывного запуска;
- минимизировать радиус поражения.

Каждый из них обсуждается по очереди в следующих главах.

Команда Netflix установила свой флаг на новой территории. Теперь они знали, что такое хаос-инжиниринг, как он работает и какую ценность представляет для большой компании.

Рождение сообщества

Как мы уже говорили, Netflix нанимал только старших специалистов. Иными словами, если вы хотите нанять хаос-инженеров, вам нужна группа опытных людей в этой области, из которой можно выбирать кандидатов. Но если вы только что создали новую дисциплину, вам трудно найти готовых специалистов. Негде было взять старших хаос-инженеров, потому что не было младших, – ведь за пределами Netflix их вообще не было.

Чтобы решить эту проблему, Кейси Розенталь решил действовать с присущим Netflix размахом и создать сообщество специалистов с нуля. Он начал с того, что осенью 2015 года организовал закрытую конференцию под названием «День сообщества хаоса». Она проходила в офисе Uber в Сан-Франциско, и в ней приняли участие около 40 человек. Были представлены следующие компании: Netflix, Google, Amazon, Microsoft, Facebook, DropBox, WalmartLabs, Yahoo!, LinkedIn, Uber, UCSC, Visa, AT&T, NewRelic, HashiCorp, PagerDuty и Basho.

Официальных тем для докладов не назначали, так что гости могли свободно говорить о проблемах, которые у них были, чтобы убедить руководство принять новую практику, а также обсуждать «провалы» и проблемы в неофициальном порядке. Организатор заранее выбрал докладчиков, способных рассказать о том, как они подошли к вопросам устойчивости, внедрения искусственных сбоев, тестирования аварийного восстановления и других методов, связанных с хаос-инжинирингом.

Учреждая «День сообщества хаоса», Netflix стремился подтолкнуть другие компании к созданию новой должности инженера по хаосу у них в штате. Это сработало. В следующем году «День сообщества хаоса» провели в Сиэтле в офисной башне Amazon Blackfoot. Менеджер из Amazon объявил, что после первого «Дня сообщества хаоса» они вернулись и убедили руководство создать команду хаос-инженеров в Amazon. Другие компании также ввели у себя должность хаос-инженера.

В том же 2016 году посещаемость выросла до 60 человек. На конференции были представлены такие компании, как Netflix, Amazon, Google, Microsoft, Visa, Uber, Dropbox, Pivotal, GitHub, UCSC, NCSU, Sandia National Labs, Thoughtworks, DevJam, ScyllaDB, C2, HERE, SendGrid, Cake Solutions, Cars.com, New Relic, Jet.com и O'Reilly.

По инициативе O'Reilly в следующем году команда Netflix опубликовала доклад на тему «Хаос-инжиниринг», который совпал с несколькими презентациями и семинаром на конференции Velocity в Сан-Хосе.

Также в 2017 году Кейси Розенталь и Нора Джонс организовали «День сообщества хаоса» в Сан-Франциско в офисе Autodesk на Market Street. (Кейси познакомился с Норой на предыдущей конференции, когда она работала в Jet.com. С тех пор она перешла в Netflix и присоединилась к команде Chaos Engineering Team.) Присутствовало более 150 человек, от завсегдатаев из крупных компаний Кремниевой долины до представителей стартапов, университетов и всего, что между ними. Это было в сентябре.

Пару месяцев спустя Нора выступила с докладом о хаос-инжиниринге на конференции AWS re:Invent в Лас-Вегасе, где 40 тысяч человек присутствовали лично и еще 20 тысяч смотрели потоковую трансляцию. Хаос-инжиниринг достиг большого успеха.

СТРЕМИТЕЛЬНАЯ ЭВОЛЮЦИЯ

Как вы увидите на протяжении всей этой книги, концепция хаос-инжиниринга стремительно развивается. Это означает, что большая часть работы, проделанной в данной области, ушла далеко в сторону от первоначальной цели. Некоторые результаты могут даже показаться противоречивыми. Важно помнить, что хаос-инжиниринг – это прагматичный подход, впервые примененный в высокопроизводительной среде, столкнувшейся с уникальными масштабными проблемами. Этот прагматизм продолжает играть ключевую роль, несмотря на то что некоторые сильные стороны современного хаос-инжиниринга основаны на науке академического уровня.

Часть

ОБЗОР ПОЛЯ ДЕЯТЕЛЬНОСТИ

На протяжении всей истории человечества сложные системы предоставляли их обладателям конкурентные преимущества. Военная наука, строительство, судоходство – люди, которым в то время приходилось работать с этими системами, сталкивались с таким количеством факторов, взаимодействующих непредсказуемым образом, что никогда не могли уверенно предсказать результат. Сегодня роль таких систем играют масштабные вычислительные системы.

Хаос-инжиниринг был задуман как дисциплина опережающего изучения и прогнозирования поведения сложных систем. В первой части этой книги представлены примеры сложных систем и обоснованы принципы хаос-инжиниринга в этом контексте. Содержание глав 1 и 2 изложено в естественном порядке, в котором опытные инженеры и архитекторы учатся управлять сложностью: размышляют, осваивают на деле, противостоят проблеме, приобретают опыт и, наконец, свободно ориентируются в этой области.

Глава 1 исследует свойства сложных систем, иллюстрируя эти свойства тремя примерами, взятыми из области вычислительных систем: «В случае сложных систем мы должны признать, что невозможно уместить все аспекты в голове одного человека». В главе 2 мы обращаем внимание на системный подход к управлению сложностью: «Целостный, системный подход хаос-инжиниринга – это одно из ключевых отличий от других практик». В главе представлены две модели для работы со сложностью: динамическая модель безопасности и экономическая модель сложности.

Глава 3 рассказывает об исследовании сложных систем, представленных в предыдущих главах, и формулирует основной принцип и определение хаос-инжиниринга: «Проведение экспериментов для выявления системных недостатков». В этой главе описана эволюция теории до настоящего момента и определены темы для последующих глав, посвященных реализации и развитию техники хаоса. «Принципы для того и разработаны, чтобы, занимаясь хаос-инжинирингом, мы знали, как это делать и как делать это хорошо».

Глава 1

Знакомьтесь: сложные системы

В первой части этой главы мы рассмотрим проблемы, возникающие при работе со сложными системами. Хаос-инжиниринг родился по необходимости в сложной распределенной вычислительной системе. В нем учтены особенности эксплуатации сложных систем, в частности присущая таким системам нелинейность, что делает их непредсказуемыми и, в свою очередь, приводит к нежелательным результатам. Это раздражает нас как инженеров, потому что нам нравится думать, что мы способны справиться с неопределенностью. Мы часто испытываем искушение винить в неполадках людей, которые строят и эксплуатируют системы, но на самом деле неприятные неожиданности являются естественным свойством сложных систем. Далее в этой главе мы спрашиваем, можем ли мы устраниить сложность системы и тем самым устраниить возможное нежелательное поведение. (Спойлер: нет, не можем.)

1.1. Размышления о сложности

Прежде чем вы сможете решить, имеет ли смысл хаос-инжиниринг для вашей системы, вам необходимо понять, где провести черту между простым и сложным. Один из способов охарактеризовать систему состоит в описании того, каким образом изменения во входных данных системы соответствуют изменениям в выходных данных. Простые системы часто называют линейными. Изменение на входе линейной системы производит пропорциональное изменение на выходе системы. Многие природные явления представляют собой линейные системы. Чем сильнее вы бросаете мяч, тем дальше он летит.

Нелинейные системы имеют выход, который может сильно меняться в зависимости от изменений в составных частях. Эффект кнута – пример системного явления¹, которое наглядно иллюстрирует нелинейность: легкое движение запястья (небольшое изменение на входе системы) приводит к тому, что дальний конец кнута разгоняется быстрее скорости звука и издает характерный громкий хлопок, которым известны кнуты (большое изменение на выходе системы).

¹ См.: Peter Senge. The Fifth Discipline. New York: Doubleday, 2006.

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru