Автоматизация доставки API

Улучшаем скорость и качество с APIOps и OpenAPI

Икенна Нвайву

Предисловие Мелиссы ван дер Хехт



MANNING

https://liveinternet.club

Automating API Delivery

APIOps with OpenAPI

Ikenna Nwaiwu
Foreword by Melissa van der Hecht



MANNING
Shelter Island

https://liveinternet.club

Автоматизация доставки API

Улучшаем скорость и качество с APIOps и OpenAPI

Икенна Нвайву

Предисловие Мелиссы ван дер Хехт

ПИТЕР®

Санкт-Петербург • Москва • Минск

2026

https://liveinternet.club

ББК 32.973.2-018
УДК 004.4+004.51
Н37

Нвайву Икенна

Н37 Автоматизация доставки API. Улучшаем скорость и качество с APIOps и OpenAPI. — СПб.: Питер, 2026. — 416 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-4256-9

Создавайте качественные согласованные API и быстро выпускайте их на рынок благодаря автоматизации разработки! Эта книга научит инновационному подходу – как применять рабочие принципы непрерывной доставки (CD) и DevOps на всем протяжении жизненного цикла API. Преобразуйте набор отдельных задач в бесшовный управляемый пайплайн с поддержкой автоматизированного тестирования, итеративных улучшений и надежной документации.

В книге представлены инструменты и стратегии, лежащие в основе APIOps. Изучите различные средства и методы улучшения процессов, такие как управление API с использованием линтера API Spectral и создание эффективного пайплайна CI/CD с помощью GitHub Actions, которые принесут быстрые и надежные результаты. Вы также научитесь использовать мощный OpenAPI Generator для автоматического создания клиентского и серверного кода на основе определений API.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018
УДК 004.4+004.51

Права на издание получены по соглашению с Manning Publications. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1633438781 англ.
ISBN 978-5-4461-4256-9

Authorized translation of the English edition © 2024 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.
© Перевод на русский язык ООО «Прогресс книга», 2026
© Издание на русском языке, оформление ООО «Прогресс книга», 2026
© Серия «Библиотека программиста», 2026

https://liveinternet.club

Краткое содержание

Глава 1. Что такое APIOps ... 31

Глава 2. Бережливый APIOps: решение проблем и внедрение методологии ... 59

Глава 3. Линтинг API: автоматизация единообразия API ... 89

Глава 4. Выявление критических изменений: управление развитием API ... 116

Глава 5. Дизайн-ревью API: проверка компонентов, недоступных для автоматизации ... 151

Глава 6. Соответствие API: генерация кода и определений API ... 185

Глава 7. Соответствие API: тестирование схем ... 215

Глава 8. CI/CD для артефактов API. 1: управление на этапе исходного кода ... 238

Глава 9. CI/CD для артефактов API. 2: стадия сборки и развертывание конфигурации API ... 280

Глава 10. Снова о соответствии API: специализированный линтинг и проверки безопасности ... 322

Глава 11. Мониторинг и аналитика: метрики продукта API ... 355

Приложение А. Значки карты потока создания ценности ... 391

Приложение Б. Установка инструментов линтинга API и выявления различий OpenAPI ... 393

Приложение В. Краткое введение в JSON Pointer ... 400

Приложение Г. Инструменты проверки соответствия API и аналитики ... 403

Приложение Д. Docker и Kubernetes ... 408

https://liveinternet.club

Оглавление

От издательства ... 18
О научном редакторе русского издания ... 18
Предисловие ... **19**
Введение ... **22**
Благодарности ... **24**
О книге ... 26
Для кого эта книга ... 26
Структура книги ... 27
О коде ... 28
Форум liveBook ... 29
Об авторе ... 30
Иллюстрация на обложке ... 30
Глава 1. Что такое APIOps ... 31
1.1. Масштабирование доставки API: возможные проблемы ... 32
1.2. Решение проблем, вызванных масштабированием ... 34
1.3. API и продукты API ... 38
1.4. Зачем внедрять APIOps? ... 41
1.5. Принципы APIOps на основе OpenAPI ... 44
1.5.1. Подход design-first для разработки API ... 44

https://liveinternet.club

1.5.2. Хранение определений API в системе контроля версий 46
1.5.3. Автоматизация соблюдения API-стандартов 47
1.5.4. Проверка API на соответствие дизайну 47
1.5.5. Хранение декларативных конфигураций шлюза API в системе контроля версий 48
1.5.6. Непрерывное согласование желаемого и фактического состояния 49
1.6. Принципы APIOps, которые вы изучите 51
1.7. Как APIOps влияет на проверку API 53
1.8. Трудности внедрения APIOps на основе OpenAPI 58
Итоги 58

Глава 2. Бережливый APIOps: решение проблем и внедрение методологии 59
2.1. Проблемы, связанные с качеством API и временем реализации 60
2.2. Метод A3 для реализации APIOps 61
2.3. Тема и предпосылки 63
2.4. Текущее состояние и постановка проблемы 65
2.4.1. Создание этапов рабочего процесса 68
2.4.2. Схема потока информации 69
2.4.3. Добавление метрик производительности 69
2.4.4. Анализ карты 71
2.5. Цель 72
2.6. Анализ первопричин 73
2.7. Контрмеры и целевое состояние 74
2.8. Подтверждение результата 76
2.9. Последующие действия 78
2.10. Ревью отчета в формате A3 79
2.11. Выявление потерь в потоке ценности 79
2.12. Преимущества использования метода A3 83
2.12.1. Развитие логического и объективного мышления 83

https://liveinternet.club

2.12.2. Достижение согласованности и налаживание совместной работы ... 85
2.12.3. Применение системного мышления ... 85
2.13. Управление внедрением APIOps ... 86
Итоги ... 88

Глава 3. Линтинг API: автоматизация единообразия API ... 89
3.1. Линтинг API в APIOps ... 90
3.2. Проблема соблюдения требований в API ... 91
3.3. Знакомство с линтерами API ... 92
3.4. Проверка на соответствие требованиям с помощью Spectral ... 93
3.5. Использование Spectral в VS Code ... 95
3.6. Создание правила Spectral для поля info.description ... 95
3.7. Создание правила Spectral для проверки версионирования путей ... 100
3.8. Подключение набора правил ... 102
3.9. Переопределение правил линтинга API ... 104
3.10. ИИ-ассистенты для написания кода ... 107
3.10.1. Промпты в Copilot для создания правил линтинга ... 109
3.11. Автоматический линтинг с помощью LLM ... 111
3.11.1. Линтинг с помощью Anthropic Claude ... 112
3.11.2. Линтинг с помощью Optic LintGPT ... 113
Итоги ... 115

Глава 4. Выявление критических изменений: управление развитием API ... 116
4.1. Изменение определений API в рабочем процессе APIOps ... 117
4.2. Проблема выявления изменений ... 117
4.3. Выявление различий в API ... 120
4.4. Обнаружение критических изменений ... 124
4.5. Генерация журнала изменений API ... 127
4.6. Критические изменения и развитие API ... 129
4.7. Схемы версионирования API ... 130

https://liveinternet.club

4.7.1. Версионирование на основе пути ... 130
4.7.2. Версионирование на основе имени хоста ... 131
4.7.3. Версионирование на основе строки запроса ... 131
4.7.4. Версионирование на основе пользовательских заголовков ... 132
4.7.5. Версионирование на основе заголовка Accept ... 132
4.7.6. Версионирование на основе привязки ... 132
4.7.7. Отсутствие схемы версионирования ... 133
4.7.8. Комбинация схем ... 134
4.8. Область версионирования API ... 134
4.8.1. Глобальное версионирование ... 135
4.8.2. Версионирование отдельных ресурсов ... 136
4.8.3. Версионирование отдельных эндпоинтов ... 137
4.9. Идентификаторы версий API ... 138
4.9.1. SemVer ... 138
4.9.2. CalVer ... 138
4.9.3. Версионирование по уровню стабильности ... 139
4.10. Затраты при переходе на новую версию ... 140
4.11. Политика работы с критическими изменениями ... 143
4.11.1. Определение критических и некритических изменений ... 144
4.11.2. Стабильность версии API ... 145
4.11.3. График выпуска версий API ... 146
4.12. Управление критическими изменениями ... 147
Итоги ... 150

Глава 5. Дизайн-ревью API: проверка компонентов, недоступных для автоматизации ... 151
5.1. Что такое дизайн-ревью API ... 152
5.2. Проведение очного ревью API ... 156
5.2.1. Назначение встречи и приглашение участников ... 156
5.2.2. Представление повестки встречи ... 158
5.2.3. Объяснение ревьюерам бизнес-контекста изменений ... 158

https://liveinternet.club

5.2.4. Проверка модели ресурсов API ... 163
5.2.5. Проверка определения API ... 167
5.2.6. Выявление новых паттернов API и расхождений с руководством по стилю ... 168
5.2.7. Завершающий опрос ... 169
5.3. Трудности ревью API ... 169
5.3.1. Принятие подхода, ориентированного на пользователя ... 169
5.3.2. Некачественное моделирование перед рецензированием ... 171
5.3.3. Слишком позднее ревью ... 171
5.3.4. Определение уровня ревью ... 171
5.4. Оценка эффективности ревью ... 172
5.4.1. Автоматизированные проверки ... 172
5.4.2. Очное ревью API ... 173
5.5. Документирование процесса дизайн-ревью API ... 174
5.5.1. Обеспечьте легкий доступ к документу ... 175
5.5.2. Сформулируйте принципы и цели ... 175
5.5.3. Опишите метрики процесса ... 176
5.5.4. Распределите роли и обязанности ... 176
5.5.5. Проводите очное ревью по запросу ... 177
5.5.6. Категоризируйте API по уровню ревью ... 177
5.5.7. Определите явный этап утверждения ... 178
5.5.8. Формализуйте внесение изменений после утверждения ... 178
5.5.9. Определите, как обозначать исключения ... 178
5.5.10. Определите способ приоритизации ревью ... 178
5.5.11. Укажите ожидаемый срок ревью ... 179
5.5.12. Обеспечьте прозрачность процесса ревью ... 179
5.5.13. Опишите, как стать ревьюером API ... 179
5.6. Примеры ревью API ... 180
5.6.1. Управление API в Microsoft Azure ... 180
5.6.2. Обработка предложений по улучшению API в Google ... 181

https://liveinternet.club

5.6.3. Дизайн-ревью API в Slack ... 182
5.6.4. Дизайн-ревью API в Kubernetes ... 183
Итоги ... 184

Глава 6. Соответствие API: генерация кода и определений API ... 185
6.1. Задача соответствия API ... 187
6.2. Генерация кода на основе OpenAPI ... 190
6.2.1. Spring Boot с OpenAPI ... 190
6.2.2. Развертывание на шлюзе ... 197
6.2.3. Публикация на портале разработчика ... 200
6.2.4. Обновление источника истины ... 201
6.2.5. Повторная публикация определения API на портале ... 204
6.2.6. Генерация клиентских SDK ... 205
6.2.7. Преимущества генерации на основе источника истины ... 207
6.2.8. Обработка ответов от шлюза ... 209
6.2.9. Настройка генерации кода ... 210
6.3. Генерация OpenAPI из кода ... 212
6.3.1. Генератор OpenAPI с использованием springdoc ... 212
6.3.2. Что нужно помнить при генерации определений ... 213
Итоги ... 214

Глава 7. Соответствие API: тестирование схем ... 215
7.1. Тесты схем ... 216
7.2. Тесты схем на основе кода ... 218
7.2.1. Написание сквозных тестов схем ... 220
7.2.2. Написание тестов схем уровня контроллера ... 222
7.3. Что необходимо учитывать при использовании тестов схем на основе кода ... 223
7.4. Генерация тестов схем ... 224
7.4.1. Использование Schemathesis для тестирования схем ... 224
7.4.2. Использование Portman для тестирования схем ... 228

https://liveinternet.club

7.5. Использование валидирующего прокси-сервера для проверки соответствия ... 230
7.6. Сравнение методов соответствия API ... 234
7.7. Отличие тестов схем от контрактных тестов, управляемых потребителем ... 235
Итоги ... 237

Глава 8. CI/CD для артефактов API. 1: управление на этапе исходного кода ... 238
8.1. Трудности совместного использования предложений API ... 242
8.2. Решение: контроль версий ... 243
8.3. Автоматическая проверка MDRC ... 244
8.4. Введение в GitHub Actions ... 247
8.5. Подготовка MDRC ... 249
8.5.1. Проверка рабочего тикета ... 250
8.5.2. Создание шаблона пул-реквеста ... 252
8.5.3. Выполнение линтинга API ... 252
8.5.4. Выполнение проверок критических изменений ... 253
8.5.5. Статический анализ текста ... 254
8.5.6. Запрос одобрения пул-реквеста от команды управления ... 255
8.6. Автоматизированный контроль управления ... 258
8.7. Сборки пул-реквеста ... 260
8.8. Политики описания пул-реквеста ... 261
8.9. Владельцы кода ... 263
8.10. Статический анализ текста ... 265
8.11. Поддержка предложений по дизайну API ... 267
8.11.1. Понимание пользователей и их требований ... 267
8.11.2. Проектирование пайплайна для быстрой обратной связи ... 267
8.11.3. Простое определение причин сбоя сборки ... 268
8.11.4. Предоставление ссылок на руководство по стилю API при сбоях линтинга ... 268

https://liveinternet.club

8.11.5. Сбор метрик времени в ходе дизайн-ревью 269
8.11.6. Включение статических проверок безопасности в сборку пул-реквеста 269
8.12. Визуализация определения API для ревью 270
8.12.1. Интеграция репозитория исходного кода с ADC 270
8.12.2. Публикация файла определений OpenAPI для пул-реквеста на сервере 272
8.12.3. Использование плагина для просмотра исходного репозитория 272
8.13. Повторное использование объектов API: мультифайловые определения API 272
8.13.1. Разбиение определения OpenAPI 274
8.13.2. Объединение мультифайловых документов 277
8.13.3. Что необходимо учитывать при работе с мультифайловыми определениями API 277
8.13.4. Инструменты для работы с мультифайловыми определениями API 278
Итоги 279

Глава 9. CI/CD для артефактов API. 2: стадия сборки и развертывание конфигурации API 280
9.1. Этап сборки: преобразование файлов определений API 282
9.1.1. Обновление кода пайплайна 284
9.1.2. Сохранение определений API в реестре API 289
9.2. Стадия сборки: симуляция API с использованием моков 292
9.2.1. Преимущества симуляции API 294
9.2.2. Типы моков API и режимы работы инструментов создания моков 294
9.2.3. Использование моков API для тестирования реальных сценариев 296
9.3. Этап сборки: генерация и хранение SDK 297
9.3.1. Преимущества предоставления SDK 298
9.3.2. Проблемы при создании SDK 298

https://liveinternet.club

9.3.3. Обновление пайплайна для генерации SDK 299
9.3.4. Хранение SDK в репозиториях артефактов 301
9.3.5. Обновление пайплайна для сохранения SDK 303
9.4. Этап развертывания и тестирования: развертывание конфигурации API-шлюза 304
9.4.1. Развертывание APIOps 305
9.4.2. Обновление конфигурации приложения 308
9.4.3. Сложности GitOps 311
9.4.4. APIOps и топологии команд 312
9.4.5. Развертывание в пайплайне из примера 313
Итоги 321

Глава 10. Снова о соответствии API: специализированный линтинг и проверки безопасности 322
10.1. Зачем нужны собственные функции 323
10.1.1. Поиск повторно используемых правил 323
10.1.2. Написание теста для правила линтинга 325
10.1.3. Написание собственного правила 328
10.1.4. Определение собственной функции 329
10.1.5. Что необходимо помнить при написании собственных функций 331
10.2. Автоматизированное ревью безопасности определений API 332
10.2.1. Введение в безопасность API 333
10.2.2. API1:2023. Нарушения авторизации на уровне объектов 336
10.2.3. API2:2023. Нарушения аутентификации 338
10.2.4. API3:2023. Нарушения авторизации на уровне свойств объектов 340
10.2.5. API4:2023. Неограниченное потребление ресурсов 343
10.2.6. API5:2023. Нарушения авторизации на уровне функций 345
10.2.7. API6:2023. Неограниченный доступ к чувствительным бизнес-потокам 345
10.2.8. API7:2023. Фальсификация запросов на стороне сервера 346
10.2.9. API8:2023. Неверная конфигурация безопасности 346

https://liveinternet.club

10.2.10. API9:2023. Некорректное управление учетом ресурсов 347
10.2.11. API10:2023. Небезопасное потребление API 348
10.2.12. Проверка трафика в шлюзе с использованием OpenAPI 349
10.2.13. Генерация определений API на основе трафика 350
10.2.14. Платформы безопасности API 351
10.3. DevSecOps 352
Итоги 354

Глава 11. Мониторинг и аналитика: метрики продукта API 355
11.1. Аналитика продукта API: бизнес-задача 356
11.1.1. Разработка продукта API как цикл PDSA 357
11.1.2. Карта пути разработчика 359
11.1.3. Метрики продукта API 361
11.1.4. Отслеживание метрик для API каталога продуктов 366
11.2. Метрики мониторинга API 382
11.2.1. Метрики RED 385
11.2.2. Ограничение частоты 386
11.2.3. Инструменты мониторинга 388
11.2.4. Оповещения и синтетический мониторинг 388
11.2.5. Индикаторы уровня обслуживания, цели уровня обслуживания и соглашения об уровне обслуживания 389
11.3. Заключение 390
Итоги 390

Приложение А. Значки карты потока создания ценности 391

Приложение Б. Установка инструментов линтинга API и выявления различий OpenAPI 393
Б.1. Установка Node.js 394
Б.2. Установка Spectral 394
Б.3. Установка Visual Studio Code 395
Б.4. Установка плагинов Spectral Visual Studio Code 395
Б.5. Установка 42Crunch OpenAPI (Swagger) Editor 395

https://liveinternet.club

Б.6. Установка jq и yq ... 396
Б.7. Установка jpp ... 397
Б.8. Установка Tufin/oasdiff ... 398
Б.9. Установка swagger-ui-cli ... 398
Б.10. GitHub Copilot ... 398
Б.11. Регистрация в Anthropic Claude ... 399

Приложение В. Краткое введение в JSON Pointer ... 400

Приложение Г. Инструменты проверки соответствия API и аналитики ... 403
Г.1. Установка и запуск Kong ... 403
Г.1.1. Установка Kong для macOS ... 403
Г.1.2. Запуск Kong через Docker ... 404
Г.2. Установка Java Development Kit и Maven ... 405
Г.3. Установка OpenAPI Generator CLI ... 405
Г.4. Установка Redocly CLI ... 405
Г.5. Установка Schemathesis ... 406
Г.6. Установка Portman ... 406
Г.7. Установка Prism ... 406
Г.8. Настройка учетной записи Moesif ... 406

Приложение Д. Docker и Kubernetes ... 408
Д.1. Знакомство с Docker ... 408
Д.2. Установка Docker ... 409
Д.3. Настройка сервисов в Docker Compose ... 409
Д.4. Введение в Kubernetes ... 410
Д.5. Установка kubectl ... 414
Д.6. Установка локального кластера Kubernetes с использованием minikube ... 414

https://liveinternet.club

Посвящается моей жене, Олучи Икодия Нвайву

https://liveinternet.club

От издательства

Мы выражаем огромную благодарность компании КРОК за помощь в работе над русскоязычным изданием книги и вклад в повышение качества переводной литературы.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

О научном редакторе русского издания

Анна Белых — старший инженер-разработчик компании КРОК. Занимается проектированием и разработкой высоконагруженных информационных систем разных масштабов. Специализируется на вопросах производительности и оптимизации серверной (backend) части.

https://liveinternet.club

Предисловие

За последние десятилетия API из разрозненных кусочков кода превратились в бизнес-продукты, и мне посчастливилось работать с их поставщиками, которые сделали возможным это превращение. Мы выработали общепринятые лучшие практики и операционные модели API и создали инструменты, которые позволили компаниям выстраивать свою деятельность на основе API. Занимая технические позиции, ориентированные на клиента, я помогала сотням глобальных организаций создавать API как часть их стратегий интеграции и наблюдала за влиянием этих проектов на людей, процессы и технологии. Я видела, как происходило становление отрасли API — это было нечто феноменальное.

Но у нас появилась проблема.

Действительно качественные API ускоряют доставку продуктов, повышают операционную эффективность и расширяют присутствие на рынке. API настолько влиятельны, что используются повсюду. Они лежат в основе всех цифровых трансформаций. Такое возможно только в том случае, если рассматривать API как продукты с полным жизненным циклом: обеспечивать последовательное и безопасное проектирование, построение и развертывание и относиться к потребителям API как к своим клиентам.

И как же резко все меняется, когда мы начинаем рассматривать API с этой точки зрения! Компании меняли курс, и те, кто полностью принял мощь API, достигали процветания. Появлялись новые целевые архитектуры, основанные на современных API. Сами API стали монетизируемыми продуктами. Чем больше компаний начинало ощущать пользу от нового подхода к разработке API, тем больше средств они вкладывали и тем быстрее старались его реализовать.

Энтузиазм, с которым многие организации старались следовать жизненному циклу API и рассматривать их как продукты, был очевиден. Принятие экономики API миллионами людей со всего мира способствовало развитию как технологий, так и бизнеса. Сейчас мы живем в цифровом мире, где API играют важную роль; мы используем их повсюду — миллионы разных API!

https://liveinternet.club

Но с принятием API в таком масштабе я заметила одну типичную проблему, которая встречалась в компаниях всех размеров, отраслей и уровней зрелости: они теряли возможность быстро поставлять API, которые всегда были бы качественными, безопасными, эффективными и последовательно представляющими свой бренд, а затем поддерживать и оценивать их влияние длительное время после развертывания.

Как достичь всего этого в сегодняшних масштабах? Как решать эти задачи еще быстрее завтра? Как делать это лучше и безопаснее? Большинство организаций еще не знают ответов на эти вопросы. Я проводила опросы в сотнях компаний, и 80 % говорили, что им приходится идти на компромисс между скоростью доставки и качеством API. Они либо выделяли время на следование жизненному циклу API для обеспечения качества, соответствия стандартам и безопасности, что замедляло достижение результата, либо ускоряли поставку за счет обходных путей в процессах управления, что приводило к повышению рисков.

Нам пока не удается извлечь реальный максимум пользы из наших API, потому что процессы, которые мы создавали для управления их жизненным циклом, не поспевали за внедрением API. В какой-то момент они стали сдерживать нас.

Хорошая новость в том, что у этой проблемы есть решение: применять и адаптировать проверенные принципы автоматизации DevOps и GitOps к жизненному циклу API в форме APIOps. Объединяя мощь линтинга, декларативной конфигурации, контроля версий и пайплайна CI/CD со своими стандартами, вы можете гарантировать, что все API будут всегда автоматически создаваться по вашим правилам.

Мои усилия в этой области вдохновляли компании на принятие методологии APIOps, и в результате сейчас они стабильно и быстро поставляют высококачественные, безопасные, объективно оцениваемые API. Однако они не должны быть единственными, кто пользуется этими преимуществами, и с помощью своей книги Икенна делает мощь и широту возможностей APIOps доступной для каждой организации.

Он мастерски проведет вас по пути изучения концепций и реальности APIOps, пока вы самостоятельно не создадите свой первый сквозной пайплайн и не поймете, *что* и *для чего* вы делали. Умело комбинируя сценарии и диаграммы и говоря на понятном языке, Икенна представит вам обзор лучших практик, но не ограничится этим. Шаг за шагом он покажет, как адаптировать их для реальных условий с примерами, практическими упражнениями и акцентом на обсуждении особенностей применения теории в конкретных условиях.

Самое трудное в применении APIOps — не только понять теорию, но и реализовать технические и культурные изменения. Уникальность книги в том, насколько четко она описывает путь к этой цели. Будь то последовательность технических этапов, примеры использования инструментов с открытым исходным кодом или эффективное проведение дизайн-ревью API, Икенна предлагает практические

https://liveinternet.club

рекомендации, а также советы из своего практического опыта по решению типичных проблем, с которыми вы можете столкнуться на этом пути.

После выхода этой книги ничто не помешает любой организации использовать APIOps, чтобы раскрыть весь потенциал API. И я с нетерпением жду, когда это произойдет!

Мелисса ван дер Хехт (Melissa van der Hecht),
консультант в стартапе API,
ранее CTO в Mulesoft и Field CTO в Kong

https://liveinternet.club

Введение

В октябре 2021 года я присоединился к 10x Banking в качестве владельца продукта в платформенной команде API. В то время компания быстро расширяла REST API для своей основной банковской платформы, что требовало не только разработки, но и управления и развертывания API. Необходимость баланса между быстротой развертывания API и соблюдением руководства по стилю API в условиях совместной работы со многими функциональными командами представляла собой непростую, но интересную задачу.

Несмотря на наши усилия по оптимизации процессов, включая усовершенствование схемы управления тикетами Jira, мне приходилось постоянно искать более эффективные решения. Поиски привели меня к замечательной презентации Мелиссы ван дер Хехт APIOps: Automating API Operations for Speed and Quality at Scale на конференции apidays в Лондоне. Ее стремление к автоматизации всего жизненного цикла API совпадало с моим, что привело к долгим поискам того, как можно адаптировать эти принципы для 10x. К январю 2022 года я разработал план актуализации API, который был хорошо принят руководством. Также я поделился этим планом с коллегами в нашем внутреннем сообществе (которое мы называли «гильдией API»), и вместе мы отправились по пути изменений.

Наша инициатива началась с анализа рабочего потока API для выявления точек ручного вмешательства и изучения возможностей для повышения их эффективности или введения автоматизации для их оптимизации. Этот путь включал замену устаревших инструментов API и интеграцию файлов OpenAPI в репозитории Git, что в конечном счете привело к значительному улучшению нашего пайплайна CI/CD (непрерывной интеграции/непрерывной доставки). Это позволило автоматизировать проверки файлов OpenAPI и развертывания в нашем API-шлюзе, в результате чего ревью API превратилось из испытания, занимавшего 2,5 недели, в задачу на один день. Такая эффективность не только повышала последовательность процесса, но и положительно влияла на нашу культуру, облегчая создание более продуманного API.

https://liveinternet.club

По мере того как организации масштабировали свои усилия по разработке API, автоматизация средств контроля и качественные инструменты разработки API начали играть определяющую роль в обеспечении надежного управления API, первоклассного опыта разработки для создателей API и оптимального взаимодействия с продуктом для потребителей API. В этой книге я постараюсь поделиться некоторыми методами, которые повысят эффективность рабочих процессов API. Кроме того, я предлагаю подписаться на мою рассылку ApiDelivery (https://ikenna.co.uk/newsletter), где я продолжаю обсуждение этих тем. Со мной также можно связаться на LinkedIn для дальнейших обсуждений и обмена мнениями об автоматизации рабочего потока API: www.linkedin.com/in/ikenna.

https://liveinternet.club

Благодарности

Я благодарю своих редакторов Тони Арритолу (Toni Arritola) и Иэна Хоу (Ian Hough), которые терпеливо учили меня стилю Manning и направляли в работе над книгой. Большое спасибо шеф-редактору издательства Брайану Сойеру (Brian Sawyer) за оказанную поддержку. Также хочу поблагодарить моих научных редакторов Дэниела Кокота (Daniel Kocot), руководителя службы API Experience and Operation в codecentric, и Мариукку Ниниоя (Marjukka Niinioja), консультанта API и соучредителя Osaango, за ценные советы, сделавшие рукопись значительно лучше. Отдельное спасибо Мариукке за тщательную проработку всех примеров в Windows и выявление многих упущенных подробностей. Кроме того, я от всей души благодарю всех сотрудников Manning, которые помогли довести работу над книгой до конца.

Благодарю компанию 10x Banking, где я трудился, за поддержку во время работы над книгой. Компания 10x предоставила среду, открытую к обсуждению и уточнению идей, представленных в книге. Моя благодарность также бывшим коллегам из 10x, включая Грэма Робинсона (Graham Robinson), Джоэла Рейха (Joel Reich), Люка Эндрюса (Luke Andrews), Данило Куджиа (Danilo Cugia), Жоао Прадо (Joao Prado) и Стюарта Коулмана (Stuart Coleman) за их организационную и техническую поддержку в преобразовании APIOps; это дало мне ценный опыт, который я отразил в книге.

Я многим обязан своим бывшим коллегам из платформенной команды API в 10x FT4. Спасибо всем: Амукте Акинпалли (Amuktha Akinpally), Сунити Дашу (Suniti Dash), Хоне Санкару (Hona Sankar), Хазу Рахаматуле (Haz Rahamathulah), Оле Афолаби (Ola Afolabi), Дхананджаю Шивагайе (Dhananjay Shivagaje), Махаву Пандею (Manav Pandey) и нашему замечательному аналитику Йониту Магарику (Yoneet Magarik) — все они помогали реализовать идеи APIOps, представленные в книге, и делились своими ценными наблюдениями и опытом.

Группа рецензентов API из 10x — Зои Верси (Zoe Versey), Люк Эндрюс, Изабель Чоудхри (Isabelle Choudhry), Рикардо Гонсалес (Ricardo Gonzalez), Акшай Неги (Akshay Negi), Йонит Магарик, Томас Крейн (Thomas Crane), Рамзи Де Костер

https://liveinternet.club

(Ramzi De Coster), Хелена Гонцарик-Пихлер (Helena Gontsarik-Pichler), Цинтия Бетеа (Cynthia Bethea), Кен Дэвис (Ken Davies), Бэйли Деви (Bailee Devey) и Валентин Нагасевши (Valentin Nagacevschi) — поделилась замечательными идеями и предложениями по автоматизации ревью и доставки API. Многие из их предложений были реализованы в книге.

Также хочу отдельно поблагодарить Ричарда Мура (Richard Moore), Данило Куджиа, Акшая Неги, Эбигейл Макферсон (Abigail MacPherson), Саймона Лофтуса (Simon Loftus) и Якуба Рожека (Jakub Rozek) за рецензирование рукописи и множество полезных предложений.

Я от всей души благодарен Эрику Уайлду (Eric Wilde) за наставления. Спасибо Вону Шерману (Vaughan Sharman) за то, что он разжег мой интерес к API во время нашей работы в 11:FS. Также хочу поблагодарить Хосе Харо Перальту (José Haro Peralta) за поддержку, и замечательного Мехди Меджауи (Mehdi Medjaoui) за помощь в продвижении книги.

Спасибо всем рецензентам, замечания которых помогли мне улучшить книгу: это Мишель Аддучи (Michele Adduci), Лауд Бентил (Laud Bentil), Джеймс Блэк (James Black), Серхио Бритос (Sergio Britos), Гонсало Уэрта Канепа (Gonzalo Huerta Cánepa), Эрнесто Карденас Кангауала (Ernesto Cárdenas Cangahuala), Прашант Двиведи (Prashant Dwivedi), Омар Эль Малак (Omar El Malak), Жиакомо Гамба (Giacomo Gamba), Люк Гринлиф (Luke Greenleaf), Асиф Икбал (Asif Iqbal), Сатиш Джупалли (Satish Jupalli), Анант Кадияла (Anant Kadiyala), Кшиштоф Камышек (Krzysztof Kamyczek), Оливер Кортен (Oliver Korten), Вивек Кришнан (Vivek Krishnan), Викер Люн (Vicker Leung), Ален Ломпо (Alain Lompo), Шрирам Мачарла (Sriram Macharla), Патрис Мальдаг (Patrice Maldague), Ник Макгиннес (Nick McGinness), Ричард Мейнсен (Richard Meinsen), Тед Моргадо (Tede Morgado), Эдер Андре Авилья Ниньо (Eder Andrés Ávila Niño, Хосе Альберто Рейес Кеведо (José Alberto Reyes Quevedo), Элиас Рангел (Elias Rangel), Конор Редмонд (Conor Redmond), Маттео Росси (Matteo Rossi), Андре Сакко (Andres Sacco), Сатеш Кумар Саху (Satej Kumar Sahu), Уильям Джамир Сильва (William Jamir Silva), Шрихари Шридхаран (Srihari Sridharan), Венкатеш Сундарамурти (Venkatesh Sundaramoorthy), Гилберто Таккари (Gilberto Taccari), Нолан То (Nolan To), Ферит Топку (Ferit Topcu), Антониос Цалтас (Antonios Tsaltas), Стефан Туральски (Stefan Turalski), Нареш Васвани (Naresh Waswani), Фил Уилкинс (Phil Wilkins) и Лукаш Витчак (Łukasz Witczak).

Наконец, огромнейшее спасибо моей жене Олучи Нвайву за ее невероятную поддержку и содействие, без которых эта книга не появилась бы на свет. Я также благодарю профессора Калу Ука (Kalu Uka) и госпожу и господина К. О. Нвайву (C. O. Nwaiwu) за их всестороннее содействие.

https://liveinternet.club

О книге

«Автоматизация доставки API» — это практическое руководство по автоматизации сквозного рабочего потока API. В книге исследуются концепции и методы реализации, необходимые для сокращения времени вывода API на рынок, уменьшения рутинной работы и улучшения опыта потребителей API. Примеры книги основаны на REST API, определенных в OpenAPI. Это издание посвящено не столько проектированию API, сколько обеспечению согласованности проектирования API, надежности генерации артефактов API и развертывания API.

Для кого эта книга

Книга предназначена для разработчиков – членов платформенных команд API, архитекторов API, а также владельцев и руководителей продуктов API. В ней рассматриваются проблемы масштабирования предложений API для организаций и быстрой поставки этих API при соблюдении стандартов управления. Так как некоторые главы в большей степени относятся к определенным ролям, я отмечаю роли, для которых материал наиболее актуален, во вводной части каждой главы. В общем случае читатель должен иметь опыт в следующих областях:

- проектирование RESTful API;
- участие в жизненном цикле разработки, включая этапы проектирования, реализации, тестирования и развертывания;
- написание базового кода сервисов и тестов, желательно на Java (глубокие знания Java необязательны);
- использование командной строки Unix/Linux и средств контроля версий через Git и GitHub.

Кроме того, будет полезно, хотя и не обязательно, иметь опыт работы с микросервисной архитектурой Docker и Kubernetes. (В книге приводится краткий обзор Docker и Kubernetes для читателей, незнакомых с этими архитектурами, чтобы все могли разобраться в примерах.)

https://liveinternet.club

Структура книги

Книга состоит из 11 глав, каждая из которых посвящена определенному аспекту APIOps.

- В главе 1 представлена методология APIOps, описываются решаемые ею задачи, управляющие принципы, ее связь с DevOps и влияние на жизненный цикл разработки ПО.
- Глава 2 переносит внимание с технологии на стратегию. В этой главе рассматривается, как внедрять преобразования APIOps, используя принципы бережливого производства. В ней представлены карты процессов для рабочих потоков создания API через визуализацию потока создания ценности, а также рассмотрены методы решения проблем с помощью метода A3, особенно в контексте организационных изменений.
- В главе 3 излагаются основы автоматизации доставки API, в частности линтинг API. Читатель познакомится с автоматизированными методами, которые проверяют, что определения API соответствуют руководствам по стилю API, принятым в организации.
- В главе 4 рассматривается управление эволюцией API и применение автоматизированных инструментов для выявления критических изменений и генерации журналов изменений API. Также здесь приводятся рекомендации по схемам версионирования REST API и установлению политики работы с критическими изменениями API.
- Глава 5 объясняет необходимость дизайн-ревью API, выходящего за рамки того, что можно автоматизировать, включая проведение ревью и измерение их эффективности.
- В главе 6 исследуется концепция соответствия API и объясняется, как сгенерировать код сервиса API на основе определения API. Также вы узнаете, как сгенерировать определения OpenAPI на основе кода и как создавать SDK для потребителей API.
- Глава 7 продолжает тему соответствия API. В ней представлены тесты схем API, которые проверяют, что поведение API соответствует опубликованному определению, с использованием тестов на основе кода и проверяющего прокси-сервера.
- Глава 8 объединяет описанные практики в пайплайн развертывания с использованием платформы GitHub Actions, при этом основное внимание уделяется управлению исходными файлами определений OpenAPI как мультифайловыми определениями.
- В главе 9 подробно изучается тема CI/CD-пайплайнов для артефактов API, включая преобразования определений OpenAPI, хранение артефактов и симуляцию API. Глава завершается темой развертывания конфигураций API-шлюзов средствами GitOps.

https://liveinternet.club

- Глава 10 возвращается к теме линтинга API. Она научит вас писать собственные функции линтинга, пользоваться линтером для обнаружения уязвимостей безопасности API OWASP и т. д.
- Глава 11 посвящена мониторингу и аналитике API. В ней представлены метрики для оценки успешности продукта API и мониторинга его технической производительности.

Книга предназначена для управления деятельностью архитекторов API и платформенных инженеров с применением практик APIOps, и ее структура предполагает последовательное чтение. Для владельцев и менеджеров продуктов API особенно актуальными могут быть главы 1, 2, 4, 5, 10 и 11, предоставляющие информацию о стратегическом планировании, безопасности и мониторинге производительности.

О коде

Книга содержит множество примеров исходного кода как в нумерованных листингах, так и в тексте. В обоих случаях исходный код форматируется моноширинным шрифтом, в отличие от обычного текста. Иногда для кода также применяется **жирный шрифт**, чтобы выделить фрагменты, изменившиеся по сравнению с предыдущими шагами, — например, при добавлении новой функциональности в существующую строку кода.

Во многих случаях оригинальная версия исходного кода переформатируется; добавляются разрывы строк и измененные отступы, чтобы код помещался на странице. Иногда даже этого оказывается недостаточно и в листинги включаются маркеры продолжения строк (➥). Также из исходного кода часто удаляются комментарии, если код описывается в тексте. Многие листинги сопровождаются аннотациями, объясняющими важные концепции.

Исполняемые фрагменты кода можно загрузить из версии liveBook (электронной) по адресу https://livebook.manning.com/book/automating-api-delivery. Полный код примеров книги доступен для загрузки на сайте Manning по адресу www.manning.com/books/automating-api-delivery. Кроме того, исходный код книги доступен по адресу https://github.com/apiopsbook/apiops. Вы можете напрямую клонировать этот репозиторий в своей учетной записи GitHub. Репозиторий кода включает файлы OpenAPI в формате YAML, конфигурации API (также в формате YAML) и примеры сервисов API, написанные на Java. Также в нем содержатся файлы конфигурации Docker и файлы манифестов Kubernetes. В примерах пайплайнов в качестве инструмента CI/CD используется платформа GitHub Actions, представленная в главе 9. Подробная информация о других необходимых инструментах и о том, как их устанавливать, приведена в приложениях Б, Г и Д.

https://liveinternet.club

Форум liveBook

Приобретая книгу «Автоматизация доставки API», вы также получаете бесплатный доступ к платформе для онлайн-чтения liveBook издательства Manning (на английском языке). Эксклюзивные возможности liveBook позволяют оставлять комментарии как к книге в целом, так и к отдельным ее разделам или абзацам. Можно легко делать заметки для себя, задавать технические вопросы и отвечать на них, а также получать помощь от авторов и других пользователей. Чтобы получить доступ к форуму, посетите страницу https://livebook.manning.com/book/automating-api-delivery/discussion. Информацию о форумах Manning и правилах поведения на них см. на https://livebook.manning.com/discussion.

В рамках своих обязательств перед читателями издательство Manning предоставляет ресурс для содержательного общения читателей и авторов. Эти обязательства не подразумевают конкретную степень участия автора, которое остается добровольным (и неоплачиваемым). Задавайте автору хорошие вопросы, чтобы он не терял интереса к происходящему! Форум и архивы обсуждений доступны на веб-сайте издательства, пока книга продолжает издаваться.

https://liveinternet.club

Об авторе

Икенна Нвайву — ведущий консультант в компании Ikenna Consulting, где он помогает организациям совершенствовать их API и процессы управления ими. Он начал свою карьеру как разработчик в Thoughtworks и с тех пор работал в разных компаниях, включая UBS, Bank of America и 10x Banking. Является обладателем ученых степеней Федерального технологического университета Оверри, Университета Шеффилда и Бизнес-школы Уорика.

Иллюстрация на обложке

На обложке книги изображен житель островов Северо-Восточной Азии (Insulaires nord Est de l'Asie). Иллюстрация взята из французской книги Жака Гассе де Сен-Совера (Jacques Grasset de Saint-Sauveur), опубликованной в 1788 году. Каждая иллюстрация этого каталога тщательно прорисована и раскрашена от руки.

В прежние времена по одежде человека можно было легко определить, где он живет и какова его профессия или положение. Manning отдает дань изобретательности и инициативности компьютерных технологий, используя для своих изданий обложки, демонстрирующие богатое вековое разнообразие региональных культур, оживающее на изображениях из собраний, подобных этому.

https://liveinternet.club

1 Что такое APIOps

В этой главе

- ✓ Проблемы, связанные с масштабированием разработки и доставки API
- ✓ APIOps как комбинация DevOps и GitOps для разработки API
- ✓ Принципы APIOps
- ✓ Преимущества и недостатки APIOps на основе OpenAPI

Во многих организациях стремятся разрабатывать качественные, хорошо документированные и простые в использовании API, однако достичь этого на практике бывает совсем непросто. Порой не помогают ни четкие стандарты и функциональные возможности управления, ни подход на основе проектирования — итоговый продукт все равно разительно отличается от задуманного. Реализованные API зачастую не соответствуют требованиям к удобству и безопасности, имеют проблемы с документацией, а пользователям трудно найти их среди других ресурсов и получить к ним доступ. Более того, могут возникать проблемы с производительностью, надежностью и масштабируемостью. Разработка API может обернуться весомыми затратами и оказаться неэффективной, из-за чего продукт появится на рынке с большим опозданием.

https://liveinternet.club

Решением проблемы служит APIOps. Этот подход помогает автоматизировать доставку API и тем самым повысить эффективность его проектирования, разработки, тестирования и развертывания. Автоматизация APIOps затрагивает управление файлами определения и конфигурации API и внедряется в пайплайн непрерывной интеграции (Continuous Integration, CI) и непрерывной доставки (Continuous Delivery, CD). APIOps обеспечивает сквозную автоматизацию жизненного цикла API, опираясь на принципы DevOps и GitOps, о которых я расскажу позже. С помощью этого подхода в компаниях разрабатывают более качественные и удобные API. На этапе проектирования традиционные методы управления, при которых человек проверяет API вручную, уступают место проверкам с помощью программных средств, включая линтинг и выявление критических изменений. Кроме того, повышается надежность релизов API, так как конфигурации развертываются через Git. API становятся более контролируемыми и лучше соответствуют требованиям, а команды разработки и эксплуатации взаимодействуют эффективнее. Таким образом благодаря тому, что в APIOps соблюдение стандартов и правил контролируется автоматически, организации могут ускорить доставку и развертывание API, а также гарантировать их согласованность. Рассмотрим это на примере.

1.1. Масштабирование доставки API: возможные проблемы

Представьте вымышленную компанию электронной коммерции, например книжный интернет-магазин BookInc. Ее ключевой продукт — облачная платформа BookStore API, которая позволяет партнерам продавать книги от имени BookInc.

На ранних этапах работы с API инженеры BookInc имели полную свободу в проектировании и развертывании изменений в REST API BookStore. Они не были связаны строгими ограничениями и поэтому могли быстро выпускать обновления. Однако с ростом компании появилось больше команд, которые вносили изменения в BookStore API, и в результате он начал работать нестабильно. Пользователи жаловались, что одни и те же ресурсы назывались по-разному в зависимости от раздела API, а сообщения об ошибках не имели единого формата. Кроме того, отсутствовала справочная документация, некоторые эндпоинты были не защищены, а другие постоянно возвращали ошибки.

Чтобы устранить эти проблемы, в компании BookInc сформировали централизованную команду для работы с изменениями BookStore API. Она разрабатывала стандарты API и следила за тем, чтобы им соответствовали все изменения, а также отвечала за работу API-шлюза и развертывание на нем API. Помимо этого, члены команды писали функциональные тесты, чтобы убедиться, что API работает в точности так, как указано в опубликованной документации.

https://liveinternet.club

В рамках управления проектированием BookStore API команда платформы вручную проверяла все предложенные изменения, чтобы убедиться, что они соответствуют стандартам. Она также обучала оптимальным методам разработки, закрепленным в стандартах API, и подчеркивала, как важно их соблюдать. Чтобы конечный продукт удовлетворял запросам пользователей, команда API-платформы призывала к новому подходу, при котором в первую очередь выполняется проектирование API (*design-first*): сначала разработчики должны создать спецификацию OpenAPI (OpenAPI Specification, OAS), пройти проверку и получить одобрение, а уже потом приступить к реализации бэкенда.

Поначалу все было хорошо. Команда API-платформы внедряла качественные решения, руководствуясь лучшими намерениями. Однако с ростом BookInc и появлением новых команд разработчиков возникло несколько проблем.

- Когда команда, разрабатывающая какую-либо функцию, хотела согласовать изменения в API, ей приходилось создавать заявки или назначать встречи со специалистами API-платформы. Так возникали задержки: несколько команд были вынуждены ждать, пока запросы попадут в план, пройдут ревью и получат одобрение. В итоге ожидание затягивалось, и на развертывание изменений в API уходило больше времени.
- Так как команда API-платформы вручную проверяла, соответствуют ли изменения стандарту, она действительно повысила согласованность API и соблюдение требований. Однако человеческий фактор внес свои коррективы: иногда возникали ошибки. Некоторые серьезные проблемы обнаруживались уже после того, как изменения попадали на продакшен и клиенты начинали использовать новую версию.
- Возросло количество развертываний API, увеличилось время, необходимое для создания и поддержки автоматизированных тестов, и нагрузка на централизованную команду API-платформы многократно повысилась. Чтобы успеть выполнять другие задачи, специалисты стали проводить ревью только один раз в неделю.
- Ревью отнимали слишком много времени, и под натиском сжатых сроков некоторые команды разработчиков отказывались от подхода *design-first* и начинали писать бэкенд-микросервисы, не дожидаясь, пока команда платформы рассмотрит предложенные изменения. Но иногда из-за этого им приходилось значительно переделывать выполненную работу, поскольку утвержденный дизайн сильно отличался от их реализации, что не могло не разочаровывать.
- Разработчики стали прилагать больше умственных усилий: чтобы не нарушать требования, им приходилось запоминать все стандарты API и следить за их обновлением.

Вот так компания BookInc изменила подход к управлению API. Отказавшись от распределенной системы, ориентированной на скорость, она перешла

https://liveinternet.club

к централизованной модели с упором на качество и надежность. Однако за это пришлось заплатить временем и снизить темпы разработки. Кроме того, процесс развертывания изменений в API стал неудобным для разработчиков: при новом подходе нужно создавать заявки, отправлять электронные письма и назначать встречи, а затем долго ждать своей очереди и медленно проходить все этапы обратной связи. Внутри компании вырос уровень напряжения и раздраженности, что плохо сказалось на корпоративной культуре.

1.2. Решение проблем, вызванных масштабированием

В нашем примере растущая компания BookInc должна сохранить баланс между качеством и скоростью выпуска API. К счастью, существует проверенный подход к разработке ПО, который позволяет справиться с такой задачей, — DevOps. Любой, кто с ним знаком, понимает, что он помогает устранить конфликты между разработчиками и командами, которые поддерживают платформу и инфраструктуру API. DevOps опирается на принципы бережливого производства (lean-принципы) и охватывает весь процесс создания ценности — от разработки до эксплуатации, — чтобы вы могли быстро и надежно доставлять ее клиентам. Одно из ключевых преимуществ подхода — ускорение работы при одновременном снижении количества отказов, то есть при повышении качества. Вы увидите, как это помогает оптимизировать процессы разработки BookInc API.

В книге «The DevOps Handbook»[1] (2-е изд., IT Revolution Press, 2021) Дж. Ким (G. Kim) выделяет три принципа DevOps.

1. Создание потока ценности с быстрым течением слева направо (от разработки к эксплуатации).
2. Налаживание потока обратной связи (с течением справа налево), который позволяет оперативно выявлять и устранять проблемы.
3. Формирование благоприятной среды для постоянного обучения, обмена знаниями и экспериментов, которая способствует доверию внутри команды.

Эти принципы ложатся в основу методов DevOps, включающих следующее.

- *Контроль версий* — использование систем для отслеживания ревизий и истории изменений кода и файлов конфигурации. Позволяет просматривать правки, восстанавливать предыдущие версии и исправлять ошибки. С помощью такой системы несколько разработчиков могут писать код совместно, сохраняя всю историю изменений. В качестве примера популярного инструмента для контроля версий можно назвать Git.

[1] Дж. Ким и др. «Руководство по DevOps. Как добиться гибкости, надежности и безопасности мирового уровня в технологических компаниях».

https://liveinternet.club

- *Непрерывная интеграция и непрерывная доставка* (CI/CD) — подход, при котором разработчики несколько раз в день интегрируют протестированный код в общую ветку репозитория. Системы непрерывной интеграции (или билд-серверы) автоматически собирают и упаковывают код в готовые артефакты, благодаря чему возможно их частое развертывание. Следующий этап — непрерывная доставка, во время которой артефакты автоматически развертываются в тестовой и продакшен-среде (иногда в нескольких). Благодаря CI/CD разработчики всегда готовы развернуть артефакты, которые прошли стандартный процесс тестирования.
- *Инфраструктура как код* (Infrastructure as Code, IaC) — способ определения системных ресурсов и топологии, например сетей, виртуальных машин и балансировщиков нагрузки (а также управления ими), с помощью описательной модели. Он позволяет изменять и хранить определения ресурсов в системе контроля версий так же, как и программный код. Благодаря таким инструментам IaC, как Terraform, Ansible или AWS, можно автоматизировать развертывание инфраструктуры, избегая ручных процессов и предотвращая дрейф окружения. IaC обеспечивает воспроизводимость и стабильность среды при каждом новом развертывании.
- *Конфигурация как код* (Configuration as Code, CaC) — подход к управлению конфигурацией приложения с помощью описательной модели, которая хранится в системе контроля версий. В отличие от подхода IaC, касающегося инфраструктуры, CaC относится к уже развернутым приложениям. Он позволяет командам создавать паттерны конфигураций, внедрять их и отслеживать изменения, предотвращая дрейф конфигурации.
- *Непрерывный мониторинг и измерения* — сбор, анализ и передача данных о состоянии системы на продакшен, благодаря которым разработчики могут ее улучшать. Этот принцип охватывает два параметра: мониторинг и наблюдаемость. *Мониторинг* включает в себя настройку метрик, оповещений и дашбордов, которые позволяют командам отслеживать состояние систем и оперативно реагировать на заранее выбранные изменения параметров. *Наблюдаемость* — это возможность запрашивать и анализировать информацию о поведении инструментальных систем и сред, чтобы выявить первопричину проблем, даже если их не удалось предвидеть. Инструменты мониторинга и наблюдаемости позволяют командам отслеживать общую производительность и состояние системы в режиме реального времени. Это достигается за счет сбора телеметрических данных, включая информацию о событиях и журналы работающих систем, — их можно анализировать и запрашивать по мере необходимости.
- *Обучение на ошибках* — в сложных программных системах неизбежно случаются сбои. Однако из них можно извлекать ценные уроки, беспристрастно анализируя инциденты и экспериментируя: иногда специалисты намеренно создают сбои, чтобы понять, как системы ведут себя в нестабильных условиях.

https://liveinternet.club

> Такая практика называется «хаос-инжиниринг». Команды устраивают «игры», в ходе которых имитируют отказ системы и проверяют свою готовность к инцидентам. Кроме того, они организуют специальные учения для тестирования безопасности, где некоторые специалисты играют роль злоумышленников из «красной команды», пытаясь осуществить взлом.

DevOps также охватывает другие принципы работы с продуктами, процессами и корпоративной культурой, включая сбор и внедрение обратной связи от пользователей, работу небольшими итерациями, упрощенные процессы утверждения, постоянное тестирование, поддержку экспериментов и взаимодействия в команде. Эффективность DevOps измеряется четырьмя основными метриками: частотой развертывания (deployment frequency), временем реализации изменений (lead time for changes), долей отказов в результате изменений (change failure rate) и средним временем восстановления (mean time to restore). Более подробно подходы и метрики DevOps описали Николь Форсгрен, Джез Хамбл и Джин Ким (Nicole Forsgren, Jez Humble, Gene Kim) в книге «Accelerate»[1] (IT Revolution, 2018).

Вернемся к компании BookInc, которая выбрала трудоемкий процесс утверждения изменений в API, и рассмотрим альтернативный, более удобный для разработчиков подход к доставке ПО, решающий возникшие проблемы, — GitOps. С его помощью можно управлять изменениями в облачных приложениях и инфраструктуре. В GitOps конфигурация системы хранится и обновляется через Git. Пайплайны доставки автоматически применяют все изменения, зафиксированные в Git, к нужным средам. При этом ручное редактирование конфигурации через пользовательский интерфейс исключено, и инженеры вносят изменения только через пул-реквесты.

Одно из преимуществ GitOps в том, что он поддерживает принцип самообслуживания. В традиционном подходе разработчики отправляют команде эксплуатации тикеты на изменение инфраструктуры, однако с появлением новых специалистов обработка тикетов становится узким местом, и централизация замедляет общую работу. Модель GitOps решает эту проблему: разработчики сами изменяют конфигурацию, отправляя пул-реквесты, которые затем проверяет и утверждает команда эксплуатации. Далее процесс развертывания в нужной среде выполняется автоматически. Такой подход гораздо эффективнее и удобнее для разработчиков, поскольку основан на Git — инструменте, с которым большинство из них уже знакомы. GitOps помогает найти баланс между скоростью разработки и контролем изменений в инфраструктуре и конфигурации. А отменить обновление можно всего одной командой `git revert`.

---

[1] Н. Форсгрен, Дж. Хамбл и Дж. Ким. «Ускоряйся! Наука DevOps. Как создавать и масштабировать высокопроизводительные цифровые организации».

https://liveinternet.club

Еще одно важное преимущество GitOps — встроенные механизмы для тестирования на соответствие и возможность аудита. Все файлы конфигурации хранятся в системе контроля версий, что позволяет отслеживать, кто, когда и какие изменения внес. Специалисты также могут автоматически проверять, соответствуют ли файлы стандартам и требованиям организации, а затем формировать об этом отчеты.

Рабочая группа GitOps (https://opengitops.dev) отмечает, что в основе этого подхода лежат четыре принципа. Состояние системы (совокупность всех параметров конфигурации, необходимых для воссоздания системы) должно соответствовать следующим требованиям.

- *Декларативность*. Для каждой среды нужно описать желаемое состояние программной системы — конфигурацию приложения и инфраструктуру — в декларативной форме. При этом не указывается, как именно оно будет достигнуто. Состояние задается не набором команд или пошаговых инструкций, а в виде конечной конфигурации.
- *Версионность и неизменяемость*. Желаемое состояние хранится в системе контроля версий, например Git. Так обеспечивается его неизменяемость и сохраняется история версий.
- *Автоматическое извлечение*. Программные агенты автоматически извлекают желаемое состояние системы из выбранного источника, сравнивают его с развернутой конфигурацией и предупреждают в случае расхождения. Они обеспечивают цикл управления и обратной связи, что позволяет системе восстанавливаться самостоятельно.
- *Непрерывное согласование*. Обнаружив расхождения между желаемым и фактическим состоянием системы, программные агенты автоматически вносят корректировки. Такое случается, если специалисты изменяют состояния вручную. Какой бы ни была причина расхождения, программные агенты его выявляют и изменяют фактическое состояние так, чтобы оно соответствовало желаемому.

Более подробно о принципах GitOps в контексте облачной среды рассказали Билли Юэн, Александр Матюшенцев, Тодд Экенстам и Джесси Суэн (Billy Yuen, Alexander Matyushentsev, Todd Ekenstam, Jesse Suen) в книге «GitOps and Kubernetes» (Manning, 2021; https://mng.bz/Bdq0).

Интеграция описанных принципов DevOps и GitOps в процессы разработки и доставки API формирует концепцию APIOps — сквозную автоматизацию жизненного цикла API. Именно поэтому я упоминал, что APIOps повышает эффективность проектирования и доставки API за счет автоматизации всех этапов работы. В этой книге я рассматриваю APIOps в контексте продуктов API, разрабатываемых по принципам REST (REST API) и применяемых в микросервисных архитектурах. Для определения этих интерфейсов используется OpenAPI.

https://liveinternet.club

Применим ли APIOps к другим форматам спецификаций API?

Спецификация OpenAPI (OpenAPI Specification, OAS; www.openapis.org), ранее известная как Swagger, представляет собой язык описания интерфейсов (Interface Definition Language, IDL), предназначенный для определения и документирования API, работающих по протоколу HTTP (HTTP API). Однако OpenAPI — далеко не единственный формат, который применяется в разработке API. Рассмотрим несколько альтернатив.

- *RAML* (RESTful API Modeling Language, язык моделирования REST API; https://raml.org) — еще один язык описания интерфейсов для HTTP API. API, определенные с использованием RAML, создаются в формате YAML.
- *API Blueprint* (https://apiblueprint.org) — язык определения API, основанный на синтаксисе Markdown.
- *AsyncAPI* (www.asyncapi.com) — спецификация, созданная по образцу OpenAPI. Этот формат описания интерфейсов служит для определения и документирования API, которые работают на основе обмена сообщениями. Он не привязан к конкретному протоколу обмена сообщениями и поддерживает в том числе AMQP (Advanced Message Queuing Protocol), MQTT (Message Queuing Telemetry Transport), WebSockets, Kafka и STOMP (Simple Text Oriented Messaging Protocol).
- *gRPC* (https://grpc.io/) — высокопроизводительный фреймворк для удаленного вызова процедур (Remote Procedure Call, RCP), с помощью которого приложения и сервисы обмениваются информацией. Обычно gRPC применяется вместе с Protocol Buffers (Protobuf) — протоколом сериализации структурированных данных, не зависящим от языка программирования.

К некоторым из перечисленных форматов API, безусловно, применимы отдельные принципы APIOps, рассмотренные в этой книге. Например, с помощью инструмента для линтинга API, который мы рассмотрим в главе 2, можно автоматически проверять, соответствуют ли файлы определения AsyncAPI руководству по стилю. Однако, несмотря на возможность применить APIOps к другим форматам, подробный разбор этой темы выходит за рамки моей книги.

Я расскажу, как применять принципы APIOps на каждом этапе жизненного цикла API: автоматизировать управление файлами определения API, тестировать их на соответствие и настраивать конфигурацию API-шлюзов. Такой подход позволит ускорить доставку API клиентам, сократив время на реализацию, и при этом соблюсти стандарты управления и качества (рис. 1.1).

Итак, APIOps предполагает полную автоматизацию жизненного цикла API, но для начала важно понять, что такое продукт API и чем он отличается от интеграционного сервиса API.

1.3. API и продукты API

Программный интерфейс приложений (Application Programming Interface, API) — это набор функций и протоколов, с помощью которых программные системы взаимодействуют друг с другом. Он служит своего рода границей, через которую

https://liveinternet.club

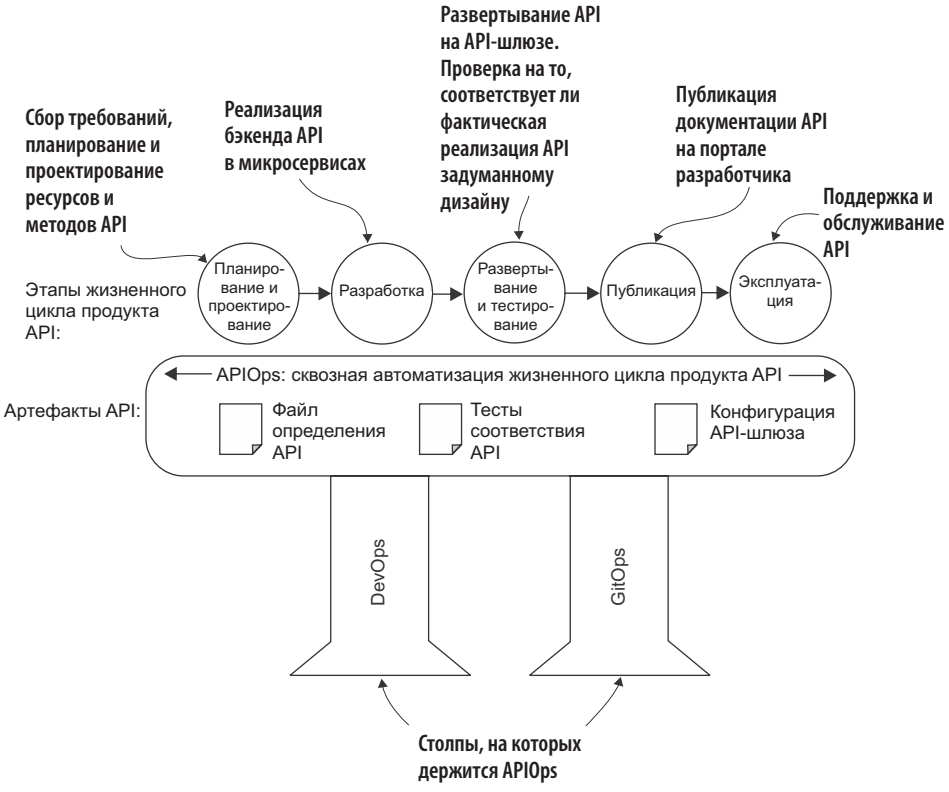


Рис. 1.1. APOps обеспечивает сквозное управление жизненным циклом API, позволяя ускорить поток создания ценности и при этом сохранить высокие стандарты управления и качества

происходит передача информации. Существуют различные типы API-решений, например REST API, SOAP API, RPC API, GraphQL API и WebSocket API. В свою очередь, *продукт API* — полноценное пакетное API-решение, ориентированное на конкретные задачи потребителя. Оно предлагает надежную систему безопасности, надлежащую документацию и каналы поддержки. За продукт API отвечает специальная команда, которая контролирует весь его жизненный цикл, от идеи до завершения поддержки. Разработчики должны понимать потребности пользователей API, улучшать взаимодействие с продуктом и следить за достижением основных метрик компании. Если продукт API предлагается на платной основе, необходимо продумать модель ценообразования и подробно объяснить, как рассчитывается и взимается плата за использование услуги. Кроме того, у продукта API должно быть соглашение об уровне обслуживания, в котором прописаны параметры качества, например время безотказной работы и ограничение частоты запросов. Девять ключевых компонентов продукта API представлены на рис. 1.2.

https://liveinternet.club



Рис. 1.2. Продукт API — комплексное решение, созданное для удовлетворения потребностей пользователя API. Он состоит из девяти компонентов, включая надлежащую документацию, каналы поддержки, а также команду, которая отвечает за его жизненный цикл

Продукты API делятся на два типа: внешние API, реализующие бизнес-функции, и внутренние, предназначенные для технического взаимодействия (например, внутренний API для управления правами доступа). API становится продуктом, когда им управляют как продуктом. Сначала разработчики выявляют и анализируют проблемы потребителей и определяют, какую пользу может принести API. Затем команда создает и утверждает дизайн API для удовлетворения потребностей, внедряет его, документирует и публикует готовое решение. Но на этом работа не заканчивается: специалисты постоянно отслеживают состояние API, анализируя отзывы пользователей и собираемые данные, и улучшают продукт на протяжении всего его жизненного цикла.

В результате API разрабатывается как самостоятельный продукт и, в отличие от обычных API-решений, требует больше ресурсов на создание документации, поддержку и заботу об удобстве пользователей. Принципы и методы APIOps, рассматриваемые к этой книге, направлены именно на разработку продуктов API.

В управлении продуктами используется модель жизненного цикла, которая помогает анализировать состояние продукта и принимать связанные с ним решения. Она описывает последовательные этапы существования продукта, включая разработку, рост, зрелость и спад. Зная их, менеджеры могут продумывать и обсуждать планирование, реализацию и развитие решений. Продукты

https://liveinternet.club

API проходят тот же самый цикл — от идеи до завершения поддержки. Чтобы эффективнее планировать, разрабатывать и поддерживать API, специалисты команды должны использовать единые термины для описания жизненного цикла. Однако каждая организация определяет его по-своему. В этой книге рассматривается упрощенная модель.

1. *Планирование и проектирование*. На этом этапе рождаются и фиксируются новые идеи для API. Определяются и анализируются функциональные и нефункциональные требования, создается концептуальная модель предметной области. Затем требования преобразуют в формальное определение API, которое компания обсуждает и утверждает с потенциальными пользователями.
2. *Разработка*. API реализуется как часть бэкенд-микросервисов. Этап включает написание кода, юнит-тестирование и упаковку. Кроме того, создается имитация (мок) API, что позволяет параллельно разрабатывать сервисы, которые будут использовать этот интерфейс.
3. *Развертывание и тестирование*. Микросервисы настраиваются в API-шлюзе, после чего API развертывается во вспомогательной среде для приемочного тестирования.
4. *Публикация*. На портале разработчика размещают документацию по API и примечания к релизу.
5. *Эксплуатация*. API развертывают и запускают на продакшене. Команда постоянно отслеживает его состояние на предмет проблем с функциональностью, производительностью и безопасностью.

Любая концептуальная модель упрощает сложные процессы, чтобы подчеркнуть в них определенные аспекты, — и эта модель не исключение. С ее помощью я хотел бы объяснить методы APIOps и подходы к управлению продуктами API на основных этапах жизненного цикла. Однако она не описывает все этапы проектирования и доставки API и не углубляется в технические подробности, поскольку обычно рабочий процесс бывает более гибким и итеративным. Если вы хотите лучше изучить жизненный цикл API, рекомендую книгу Луиса Вейра (Luis Weir) «Enterprise API Management» (Packt Publishing, 2019).

1.4. Зачем внедрять APIOps?

APIOps позволяет автоматически развертывать продукты API и управлять ими. Однако прежде чем внедрять APIOps, важно разобраться в существующем процессе создания ценности: понять ключевые этапы и действия на пути от запроса клиента до развертывания API в продакшен-среде. Эти знания помогут определить приоритетные области, выигрывающие от автоматизации. Подробнее о том, как планировать проектирование и доставку API, я расскажу в главе 2.

https://liveinternet.club

В этой книге я рассматриваю APIOps в контексте разработки REST API для микросервисных архитектур. Работа над продуктами API в микросервисных проектах — циклический процесс, который требует тесного сотрудничества разных специалистов: разработчиков, архитекторов, менеджеров по продукту, инженеров по безопасности, технических писателей, команд эксплуатации и не только. Цель APIOps — наладить эффективное взаимодействие между командами на этапах проектирования, разработки и эксплуатации API, а также автоматизировать процессы, чтобы уменьшить количество ручных действий, ускорить выполнение задач и снизить нагрузку на сотрудников.

Чтобы лучше понять процесс разработки и управления, рассмотрим компоненты микросервисной архитектуры. Она представляет собой совокупность небольших, слабо связанных между собой сервисов, каждый из которых реализует одну бизнес-функцию в рамках ограниченного контекста. Каждым микросервисом управляет небольшая команда: она создает отдельную базу кода и отвечает за хранение данных. Такой подход позволяет развертывать микросервисы независимо друг от друга. API микросервисов четко определены и могут быть сгруппированы в продукты API, которые затем станут доступны для внешнего использования через API-шлюз.

> **ПРИМЕЧАНИЕ** Ограниченный контекст (bounded context) — понятие из предметно-ориентированного проектирования (Domain-Driven Design, DDD). Оно обозначает модель предметной области внутри более крупного приложения. Это граница, в пределах которой единообразно применяются определенные термины и правила.

API-шлюз — обратный прокси-сервер, который служит точкой входа в продукты API. Он не только направляет запросы от продуктов API к API бэкенд-микросервисов, но и обеспечивает защиту, управляет трафиком, собирает аналитику, трансформирует сообщения и предлагает возможности оркестрации. API-шлюз предоставляет интерфейс продукта API его потребителям — разработчикам, создающим прикладное ПО, которое интегрируется с API. Для упрощения работы с API используется портал разработчика, где публикуется документация: справочники, руководства пользователя и обучающие материалы (рис. 1.3).

Спецификация API, определение API и Swagger

В этой книге под *спецификацией API* (API specification) я подразумеваю спецификацию OpenAPI (OAS) — общепринятый стандарт для определения REST API, понятный как людям, так и машинам. Его последнюю версию можно найти на сайте https://spec.openapis.org/oas/latest.html. На момент написания книги актуальна версия 3.1.0. Поддержку стандарта осуществляет консорциум отраслевых экспертов OpenAPI Initiative (OAI), входящий в состав Linux Foundation.

Используя термин *определение API* (API definition) или *определение OpenAPI*, я имею в виду файл с описанием конкретного API в формате OAS. Обычно это файлы YAML

https://liveinternet.club

или JSON. Другими словами, используя OAS для описания API, вы создаете файл определения. Он играет роль контракта, в котором подробно описаны поддерживаемые операции, механизмы безопасности, а также ожидаемые запросы и возможные ответы для каждой операции.

Благодаря широкому распространению стандарта появилось множество инструментов, способных читать файлы определения OpenAPI. Среди них — текстовые и графические редакторы, генераторы SDK, мок-серверы и инструменты для документирования. Более полный список можно найти на сайте http://OpenAPI.tools.

Как я уже упоминал, раньше спецификация OAS называлась Swagger Specification. В ноябре 2015 года компания SmartBear Software передала спецификацию в OAI, где ее переименовали в OpenAPI Specification. Первой версией после смены названия стала OpenAPI 3.0. Теперь Swagger — это набор инструментов SmartBear для работы со спецификацией, как платных, так и бесплатных с открытым исходным кодом, включая Swagger Editor, Swagger UI, Swagger Codegen и SwaggerHub.

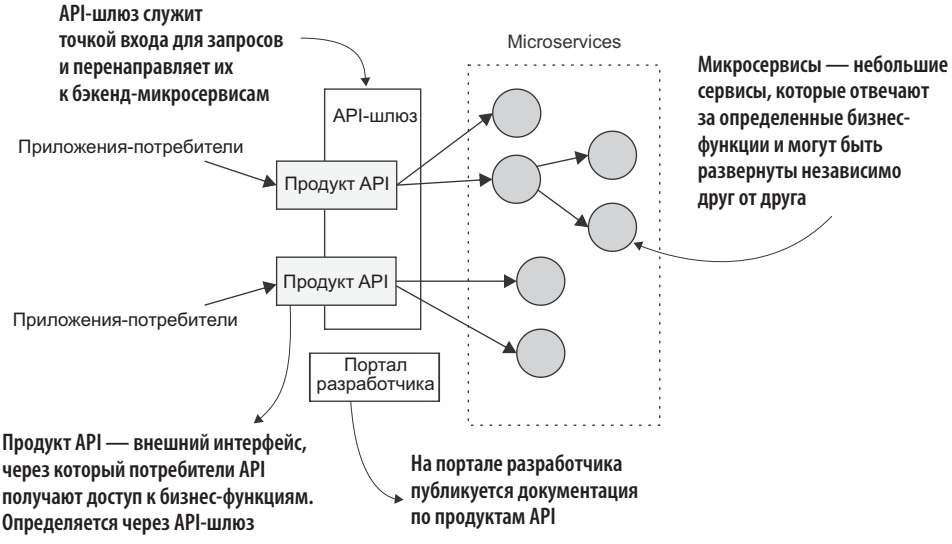


Рис. 1.3. В микросервисной архитектуре API-шлюзы определяют продукты API и направляют запросы к бэкенд-микросервисам, которые отвечают за реализацию бизнес-функций

К каждому продукту API прилагается определение — формальное описание операций и ресурсов, доступных его потребителям. API-шлюз также содержит конфигурацию API, которая определяет, как направлять внешние запросы во внутренние микросервисы. Кроме того, в ней задаются дополнительные ограничения, накладываемые на интерфейс, например механизмы безопасности и ограничение частоты запросов.

https://liveinternet.club

Таким образом, в основе APIOps, рассматриваемого в этой книге, лежат два ключевых файла: файл определения API и файл конфигурации API-шлюза. На протяжении всего жизненного цикла продукта API над ними работают разные специалисты: они создают, обновляют, проверяют, развертывают и публикуют эти файлы, а APIOps помогает работать эффективнее и соблюдать стандарты качества и комплаенс (соответствие нормативным требованиям).

1.5. Принципы APIOps на основе OpenAPI

В этой книге я покажу, как обеспечить сквозную автоматизацию разработки продуктов API, используя OpenAPI в качестве языка для определения API. Исходя из этой цели, выделим шесть принципов APIOps на основе OpenAPI.

- При разработке API используйте подход design-first.
- Храните файлы определения API в системе контроля версий.
- Автоматизируйте процесс соблюдения API-стандартов.
- Проверяйте, соответствует ли фактическая реализация API утвержденному дизайну.
- Храните декларативные конфигурации API в системе контроля версий.
- Регулярно согласовывайте фактическую конфигурацию API-шлюза с желаемой.

1.5.1. Подход design-first для разработки API

Работа с API как с продуктом, описанная в разделе 1.3, предполагает, что перед написанием кода команда проектирует API и обсуждает дизайн с коллегами и потенциальными потребителями. Процесс схож с управлением продуктом, когда сначала создают прототип и проверяют, удовлетворяет ли он потребности клиента. Обратившись к этой аналогии, файл определения OpenAPI можно считать точным прототипом, который подходит для написания документации и моделирования работы реального API. Созданный файл можно показать потенциальным потребителям, чтобы убедиться, что он решает их задачи и соответствует ожиданиям. В этом и заключается подход design-first для разработки API.

Обратите внимание, что на этапе проектирования продукта API нужно учитывать как его функциональные, так и нефункциональные параметры, а также метрики компании. Разработчики должны заранее определить механизмы безопасности для защиты API, сформулировать политики в отношении устаревания (deprecation) и внедрения критических изменений, а также установить ограничение частоты запросов и планы использования API.

Когда определение API создано и согласовано заранее, команды могут действовать параллельно. Разработчики бэкенда сразу приступают к реализации сервиса

https://liveinternet.club

API, тестировщики пишут функциональные тесты, а разработчики клиентских приложений создают интеграции на основе мока согласованного API. Если организовать процессы параллельно, вы сократите время реализации.

Подход design-first и API-ориентированная разработка

Поход design-first (также «сначала спецификация» (specification-first), «сначала контракт API» (contract-first)) предполагает, что сперва создается определение API, а уже затем пишется код для его реализации. Этот подход является важной частью управления API как продуктом. Следуя ему, необходимо продумать, кто будет применять API и какие задачи потребителей продукт способен решить. Прежде чем приступать к разработке, важно создать прототип и убедиться, что он действительно удовлетворяет запросам потенциальных пользователей.

Подход design-first предлагает несколько преимуществ.

- *Акцент на ценности и удобстве использования*. Команда создает API, которые действительно помогают решать задачи пользователей. Сначала специалисты анализируют цели потребителей, а затем совместно со стейкхолдерами (менеджерами по продукту, бизнес-аналитиками) разрабатывают API, соответствующий этим целям. Такой подход помогает создавать удобные продукты, которые удовлетворяют запросам пользователей, в отличие от решений, продиктованных исключительно техническими ограничениями и особенностями реализации.
- *Ранняя обратная связь*. Рассматривая API как продукт, можно использовать проверенную методику управления — прототипирование. На основе заранее подготовленного определения API можно создать документацию и моделировать работу API, получая отзывы от пользователей уже на ранних этапах. Обратная связь позволит вовремя выявить и исправить недочеты в дизайне, а значит, снизить затраты и ускорить разработку.
- *Параллельная разработка*. Когда определение API согласовано заранее, разработка может вестись сразу в нескольких направлениях: написание кода, подготовка сценариев тестирования и интеграция. Это сокращает время реализации API.
- *Повторное использование*. Подход design-first позволяет выявлять и повторно использовать уже готовые компоненты API и паттерны проектирования, включая схемы и модели ресурсов. Благодаря этому организация экономит время и сокращает расходы на внедрение.
- *Предметно-ориентированное проектирование*. Существует множество инструментов для автоматического создания серверных и клиентских заглушек (стабов), тестового кода, политик безопасности и конфигурации API на основе файлов определения. Все инструменты могут работать с единым файлом определения, что ускоряет разработку за счет автоматизации задач.

Противоположный подход — *подход на основе написания кода* (code-first; также «сначала код»). В нем процесс начинается с реализации API по требованиям компании, а затем на основе готового кода автоматически генерируется файл определения. Преимущество этого подхода — скорость: он особенно удобен, когда нужно внести небольшие изменения в API. Этот процесс более привычен для разработчиков, кото-

https://liveinternet.club

рые предпочитают сразу переходить от требований к написанию кода и не знакомы с инструментами проектирования API и файлами определения. Однако стоит упомянуть и его недостаток: он помогает обеспечить функциональность, но не учитывает удобство использования и единообразие API с точки зрения пользователей.

Иногда подход design-first путают с API-ориентированной разработкой (API-first), но это разные концепции. API-ориентированная разработка — это стратегия, при которой компания инвестирует в API, управляет ими как продуктом и делает их важнейшей частью своей экосистемы и критических бизнес-активов. Фактически этот подход предполагает, что API становятся основным интерфейсом для взаимодействия с приложением или платформой.

Определения OpenAPI вовсе не обязательно создавать вручную в YAML или JSON. Существуют удобные редакторы, например Stoplight (https://stoplight.io), Apicurio Studio (www.apicur.io/studio), Postman (www.postman.com) и др. Они предлагают интуитивно понятный графический интерфейс и помогают создать файлы определения API.

1.5.2. Хранение определений API в системе контроля версий

Файлы определений OpenAPI следует хранить в системе контроля версий, например Git, чтобы обеспечить неизменяемость данных и сохранить историю правок. Работа с Git открывает возможность аудита и позволяет автоматизировать процессы через пайплайн CI/CD. Команда разработчиков может сообща изменять файлы определения API, отправляя стандартные пул-реквесты.

Я рекомендую завести отдельный Git-репозиторий для определений API. Это упростит их поиск, унифицирует паттерны и позволит повторно использовать общие компоненты API. Git предлагает отличный механизм для отслеживания изменений и широкие возможности интеграции, благодаря которым можно встраивать файлы определений OpenAPI в пайплайн CI/CD.

> **СОВЕТ** Некоторые поставщики инструментов для совместного проектирования API позволяют хранить файлы определения на своей платформе. Однако я советую использовать Git в качестве основного хранилища и разрешить сторонним инструментам получать или обновлять размещенные в нем определения API. Такой подход открывает не только возможности автоматизации через CI/CD, но и избавляет от привязки к конкретному поставщику. При желании вы сможете использовать другие инструменты для редактирования файлов определения API.

Если хранить определения API в Git и вносить изменения через пул-реквесты, можно попросить коллег проверить их (как и пул-реквесты в любом другом коде), а также настроить автоматические проверки на соответствие требованиям. Это приводит нас к принципу автоматизации.

https://liveinternet.club

Обратите внимание, что есть и другое место для работы с файлами определения API — реестр. Реестр API — это сервер для хранения, поиска, просмотра, обновления и удаления файлов определения API. Однако я все же рекомендую Git или другую систему контроля версий, так как разработчикам удобно вносить изменения через зпул-реквесты, а такие хостинговые платформы, как GitHub и Bitbucket, открывают широкие возможности для интеграции. Некоторые из них предлагают плагины для просмотра визуализированных определений OpenAPI прямо в репозитории. Подробнее об этом я расскажу в главе 9.

1.5.3. Автоматизация соблюдения API-стандартов

Руководство по стилю API можно преобразовать в набор исполняемых правил и затем автоматически анализировать, соответствуют ли им файлы определения API. В некоторых случаях дополнительно пригодится линтер API — инструмент, обрабатывающий заданные правила и выявляющий участки в определении API, в которых они не соблюдаются. Для других проверок могут понадобиться специальные скрипты. Например, если в вашем руководстве по API запрещены критические изменения, можно использовать скрипт или инструмент, который сравнивает текущую и предыдущую версии файлов определения и выявляет такие изменения в дизайне API.

Чтобы написать руководство по созданию исполняемых правил, необходимо иметь руководство по стилю API. Разработка такого документа с нуля — сложная задача, однако в открытых источниках есть готовые примеры, которые могут стать хорошей основой для создания собственных правил.

Чтобы API отвечали стандартам вашей организации, важно автоматизировать процесс проверки. Например, закодируйте требования в виде исполняемых правил, чтобы линтер API или специальный скрипт могли анализировать определения API и выявлять несоответствия. Интеграция таких проверок в пайплайн CI/CD позволяет разработчикам быстро получать обратную связь и служит одной из основ APIOps.

Завершив автоматизированную проверку определений API на соответствие стандартам и согласовав дизайн со стейкхолдерами, можно переходить к реализации. Но это не значит, что определения API зафиксированы раз и навсегда. Не забывайте, что проектирование API — итеративный процесс: если определения изменятся, автоматизированная проверка поможет быстро убедиться, что API по-прежнему соответствует требованиям.

1.5.4. Проверка API на соответствие дизайну

После разработки API важно проверить, соответствует ли результат согласованному дизайну в определении OpenAPI. Для этого можно провести тестирование на соответствие — запустить специальные тесты, которые автоматически

https://liveinternet.club

создаются на основе файла определения OpenAPI. Система выполняет операции, заданные в определении, и сравнивает ответы с указанными в нем значениями, в том числе сверяет схемы объектов и выявляет нереализованные операции. Кроме того, в ходе тестирования API на соответствие может выполняться фаззинг — подача случайных, нестандартных данных, которые помогают обнаружить реакции API, не задокументированные в файле определения. Успешное прохождение тестов подтверждает, что файл определения точно описывает фактическую работу API.

1.5.5. Хранение декларативных конфигураций шлюза API в системе контроля версий

После разработки бэкенд-сервисов API необходимо развернуть их в качестве продуктов в API-шлюзе, чтобы открыть доступ клиентам. Для этого создайте или обновите конфигурацию развертывания шлюза. Она должна быть представлена в декларативном формате и храниться в Git-репозитории соответствующей среды. Все изменения конфигурации API должны происходить через Git, а не вручную через интерфейс API-шлюза. В конфигурации API-шлюза указывается различная информация: данные бэкенд-сервисов, к которым следует направлять запросы от продуктов API, механизмы идентификации и аутентификации клиентов, план использования и ограничение частоты запросов для каждого клиента, способы регистрации и мониторинга запросов и многое другое. Все параметры должны быть заданы в декларативном формате и описывать желаемое состояние API для каждой среды.

> **ПРИМЕЧАНИЕ** *Декларативный* подход позволяет описывать в файле конфигурации ожидаемое состояние системы, например ее настройки и свойства, указывая только значения параметров. Специальный агент или программный инструмент автоматически считывает этот файл и выполняет команды, которые настраивают систему в соответствии с заданным состоянием. При этом вы можете даже не знать тип и порядок вызова команд — агент самостоятельно обеспечит предсказуемый результат. Если же следовать *императивному* подходу, придется явно задать все команды и их последовательность. После запуска скрипта система выполнит все указанные действия для достижения желаемого состояния системы.

Если конфигурация API хранится в системе контроля версий (например, Git) в виде неизменяемых версионированных файлов, специалисты могут работать над ними совместно, следуя процессам репозитория. Такой метод управления объединяет подходы «конфигурация как код» и GitOps, о которых я упоминал в разделе 1.2.

Существует несколько способов задать конфигурацию для развертывания API-шлюзов. В некоторых шлюзах, например Amazon API Gateway и Kusk API Gateway, она задается в декларативном формате в определениях API в виде расширений поставщика OpenAPI. В других системах, включая Apigee Edge,

https://liveinternet.club

декларативная конфигурация хранится в отдельных файлах. В свою очередь, API-шлюзы, которые работают в Kubernetes в качестве Ingress-контроллеров, используют для конфигурации специальные пользовательские ресурсы (Custom Resource Definitions, CRD), хранящиеся в файлах манифестов. Среди таких решений можно отметить контроллеры Kong для Kong Gateway и Tyk Operator для Tyk Gateway. Но как бы ни задавалась декларативная конфигурация, для эффективной реализации APIOps она должна храниться в системе контроля версий.

1.5.6. Непрерывное согласование желаемого и фактического состояния

Чтобы реализовать последний принцип APIOps, понадобятся программные агенты. Они постоянно сравнивают фактическое состояние API-шлюза с желаемой конфигурацией, которая хранится в репозитории среды. Дрейф (расхождение между желаемым и фактическим состоянием) может возникнуть, если фактическое состояние системы было изменено напрямую (например, через административный интерфейс API-шлюза) или потому, что желаемая конфигурация в репозитории была изменена намеренно. Независимо от того, как возник дрейф, программные агенты обнаруживают его и приводят фактическое состояние в соответствие с желаемым. Эта непрерывная синхронизация желаемого и фактического состояний является важным принципом GitOps.

Программные агенты (например, контроллеры Kubernetes) автоматизируют этапы работы с развернутыми приложениями: создание, настройку и управление. Они выявляют любые расхождения между желаемой конфигурацией в репозитории и развернутым состоянием системы в рабочей среде, а затем могут предупредить команду о проблеме и автоматически исправить ее, развернув нужное состояние. Таким образом, агенты обеспечивают контур управления с обратной связью, что позволяет системе восстанавливаться самостоятельно.

Специалистам запрещено обновлять конфигурацию API-шлюза вручную через интерфейс, и все изменения должны вноситься через пул-реквесты в Git. Поскольку файлы конфигурации хранятся в репозитории, участники команды могут проводить аудит. Любой из них увидит, кто, когда и как изменил состояние системы. А если подключены программные агенты, реализующие автоматическое развертывание, отменить изменение конфигурации можно с помощью простой команды `git revert`. Подход GitOps проиллюстрирован на рис. 1.4.

Внедрение APIOps начинается с первого принципа: необходимо рассматривать API как продукт и соответствующим образом создавать и обновлять его файлы определения. Далее следует принцип версионирования, в соответствии с которым файлы определения API хранятся в Git для поддержки CI/CD-автоматизации. Все изученные принципы наслаиваются друг на друга, вплоть до последнего — автоматического развертывания конфигурации API с помощью программных

https://liveinternet.club

агентов. То, как развиваются процессы, необходимые организации для внедрения этих принципов, можно представить в виде пирамиды зрелости APIOps (рис. 1.5).

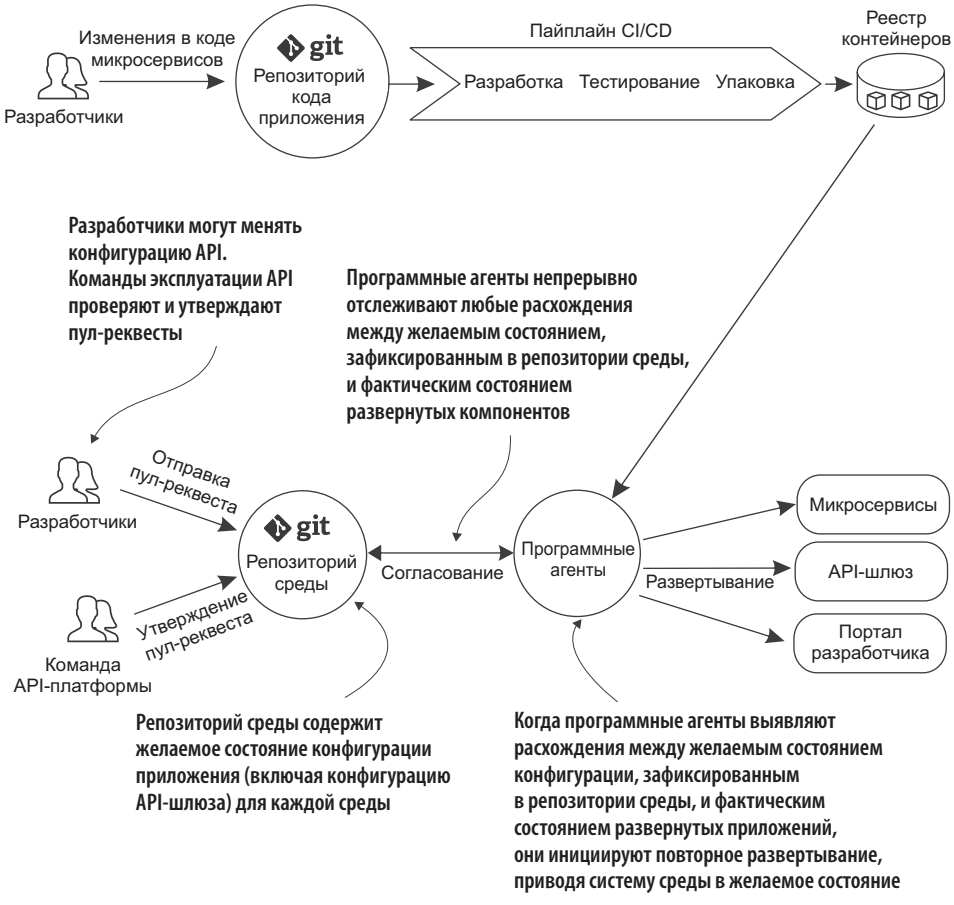


Рис. 1.4. Подход APIOps основан на принципах GitOps. Желаемая конфигурация API-шлюза и состояние сервисов описываются декларативно и хранятся в Git-репозитории среды. Разработчики вносят все изменения через Git, а команды эксплуатации проверяют и утверждают их. Программные агенты автоматически выявляют различия между фактическим состоянием и тем, что описано в репозитории. Обнаружив дрейф, агенты устраняют его, самостоятельно развертывая желаемое состояние в заданной среде

Изложенные принципы определяют работу всех специалистов, которые участвуют в создании и поддержке продукта API. В разделе 1.7 я расскажу о том, как APIOps влияет на рабочие процессы разработчиков, менеджеров по эксплуатации и менеджеров продуктов API.

https://liveinternet.club

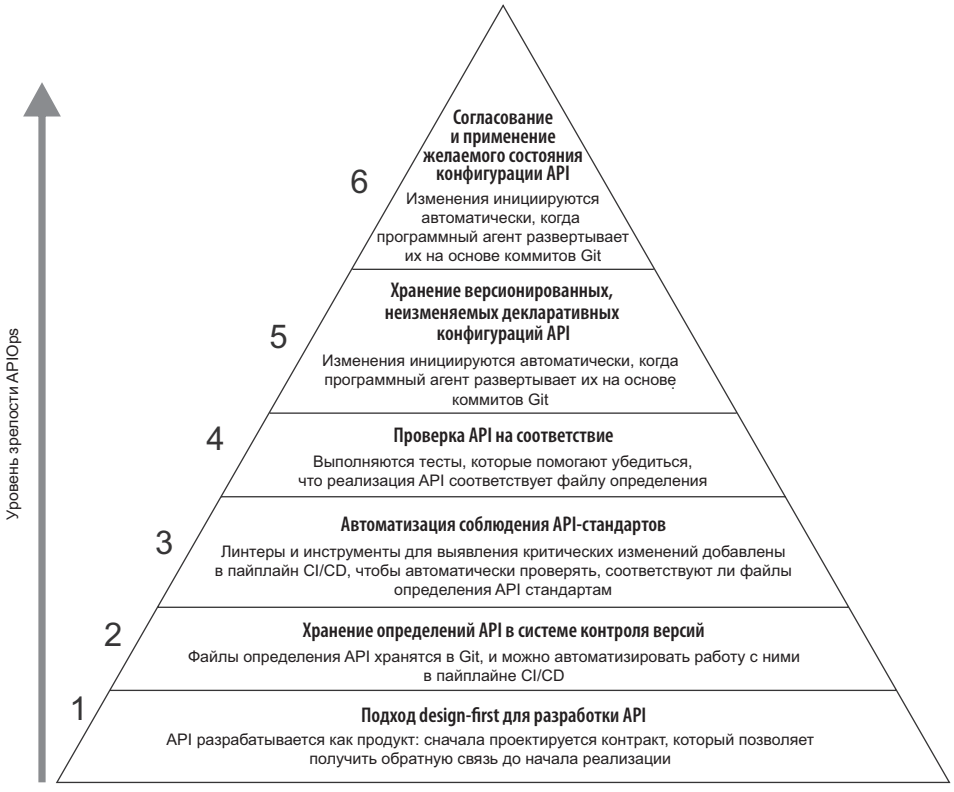


Рис. 1.5. Пирамида зрелости иллюстрирует шесть принципов APIOps, помогая организациям оценить прогресс внедрения этого подхода

1.6. Принципы APIOps, которые вы изучите

В этой книге я расскажу, как применять шесть принципов APIOps на разных этапах жизненного цикла API, изображенных на рис. 1.1. Мы рассмотрим следующие темы.

- Линтинг API.
- Автоматическое выявление критических изменений.
- Разработка и проведение тестов на соответствие.
- Генерация кода для серверной и клиентской частей на основе файлов определения API.
- Моки файлов определения API для параллельной разработки.
- Создание и развертывание конфигурации API-шлюза в соответствии с декларативным подходом.

https://liveinternet.club

- Автоматическая публикация документации API на портале разработчика.
- Генерация метаданных релиза API на основе файлов конфигурации и определения.
- Настройка мониторинга и аналитики для отслеживания работы API.

Изучая этап проектирования продукта API, вы узнаете, как создавать и применять правила линтинга к файлам определения API, чтобы дизайн соответствовал требованиям. Благодаря автоматизации процесса поймете, на чем команде лучше сосредоточиться во время ручных проверок. Помимо этого, вы узнаете, как с помощью программных средств выявлять критические изменения между различными версиями файлов определения API, и научитесь тестировать API на соответствие, чтобы убедиться, что результат совпадает с задуманным дизайном. Наконец, я расскажу, как интегрировать линтинг API и поиск критических изменений в пайплайн CI/CD.

Перейдя к этапу разработки продукта API, я научу вас генерировать серверные заглушки (стабы) и код клиентской части на основе файлов определения. Кроме того, вы узнаете, как создать мок-файл определения OpenAPI, чтобы одновременно разрабатывать приемочные тесты и код для интеграции с API.

Говоря о развертывании и тестировании, я покажу, как создать и развернуть конфигурацию API-шлюза с помощью подхода GitOps, при котором она хранится и автоматически обновляется на основе изменений в Git. Мы также разберем, как проверять API на соответствие с помощью тестов, созданных на этапе проектирования.

Затем, на этапе публикации, вы узнаете, как размещать файлы определения API в качестве документации на портале разработчика, а также генерировать примечания к релизу, извлекая метаданные из файлов определения и конфигурации API.

Наконец, добравшись до этапа продакшен, я познакомлю вас с тем, как настроить мониторинг и аналитику в режиме реального времени. Вы также поймете, как с помощью собранных данных повысить качество и ценность API.

Рисунок 1.6 демонстрирует, как методы APIOps интегрируются в рассмотренный жизненный цикл API. На каждом этапе команда API получает обратную связь. Помните, что упрощенная модель показывает лишь основные этапы работы и циклы обратной связи. Она не описывает весь технологический процесс и не дает рекомендаций по разработке API. В реальных условиях команды быстро получают обратную связь по мере выполнения разных задач. Однако модель показывает, как встроить методы APIOps в уникальный жизненный цикл API, который применяется в вашей организации.

Вы не только познакомитесь с процессами на разных этапах жизненного цикла продукта, но и получите советы о том, как, внедряя APIOps, постепенно оптимизировать доставку API.

https://liveinternet.club

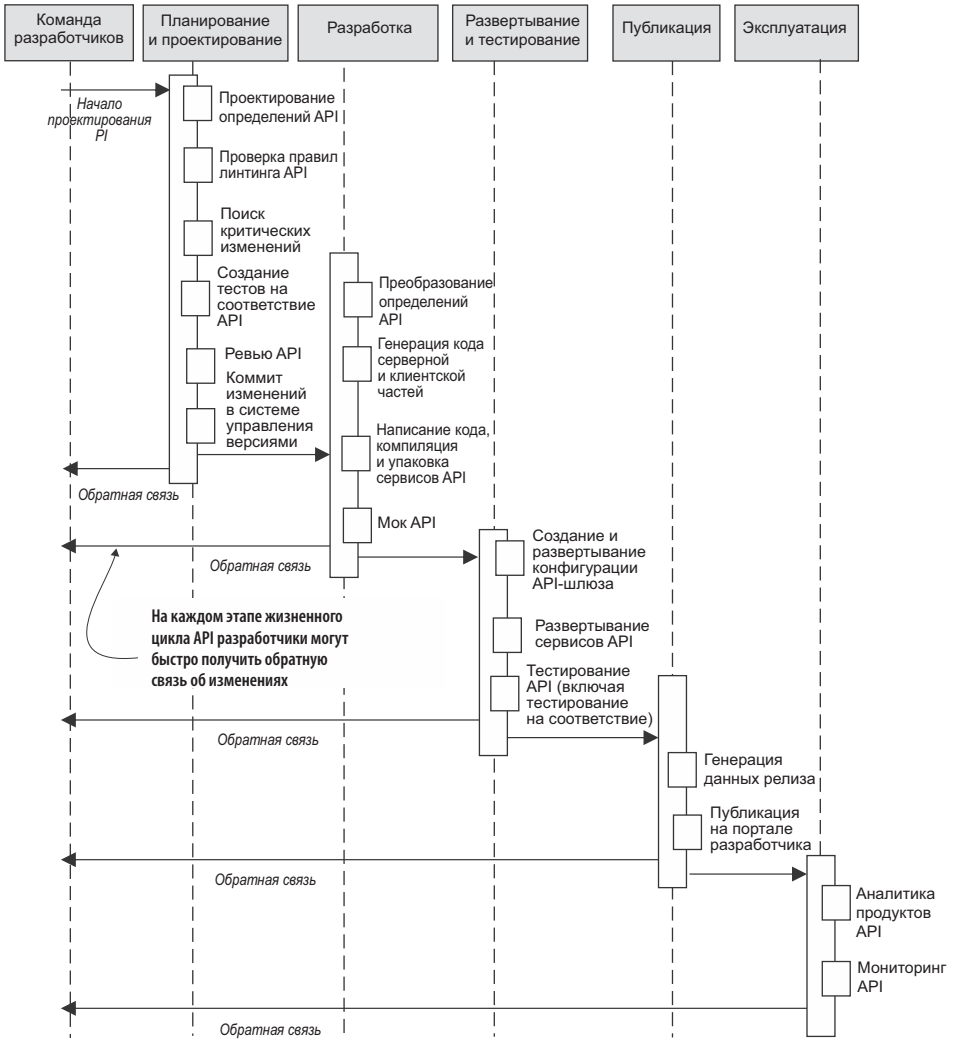


Рис. 1.6. Методы APIOps в сочетании с другими процессами на каждом этапе жизненного цикла API

1.7. Как APIOps влияет на проверку API

В предыдущих разделах я описал, как APIOps помогает организациям вывести проектирование API на новый уровень, ускорить разработку и доставку и при этом поддерживать высокий уровень единообразия, качества, надежности и безопасности API. В процессе внедрения APIOps организации формируют подход к управлению API, проходя три больших этапа.

https://liveinternet.club

На первом этапе отсутствуют централизованные стандарты API. Без единого руководства разработчикам приходится определять и применять их самим. В результате ускоряется доставка API, но возникают проблемы единообразия, ведь каждая команда следует собственным стандартам. Вместе с тем из-за возникающих несоответствий усложняется работа пользователей с API. Недостатки такого подхода проиллюстрированы на рис. 1.7. Организации, в которых отсутствуют централизованные стандарты API, поначалу работают над дизайном быстро, но по мере роста сталкиваются с проблемами: различия в дизайне и качестве API затрудняют внедрение продуктов, а также увеличивают расходы на поддержку.

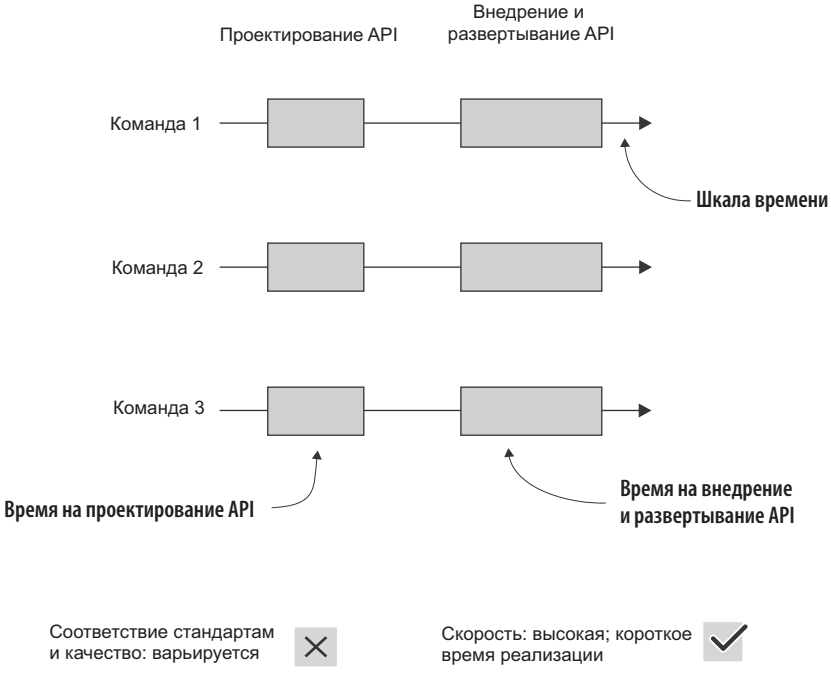


Рис. 1.7. Без централизованных стандартов и проверок на соответствие требованиям команды могут быстро доставлять API. Но в результате нарушается единообразие API и затрудняется повторное использование паттернов и компонентов

Во избежание этих проблем организации могут перейти ко второму этапу: определить стандарты API в виде руководства по стилю, которому должны следовать все команды. Кроме того, централизованная управляющая команда вручную проверяет API, чтобы убедиться, что требования действительно соблюдены. В результате создаются более единообразные, качественные API, которые соответствуют стандартам.

https://liveinternet.club

Однако централизованное ручное управление проектированием может сдерживать разработку. Доставка API замедляется, ведь командам приходится долго ждать проверок или раз за разом исправлять замечания и вновь ждать решения. Постоянные возвраты API на доработку приводят к еще бóльшим задержкам, а из-за ручных проверок в игру вступает человеческий фактор — люди могут упустить важные детали. Задержки, вызванные таким подходом, проиллюстрированы на рис. 1.8.

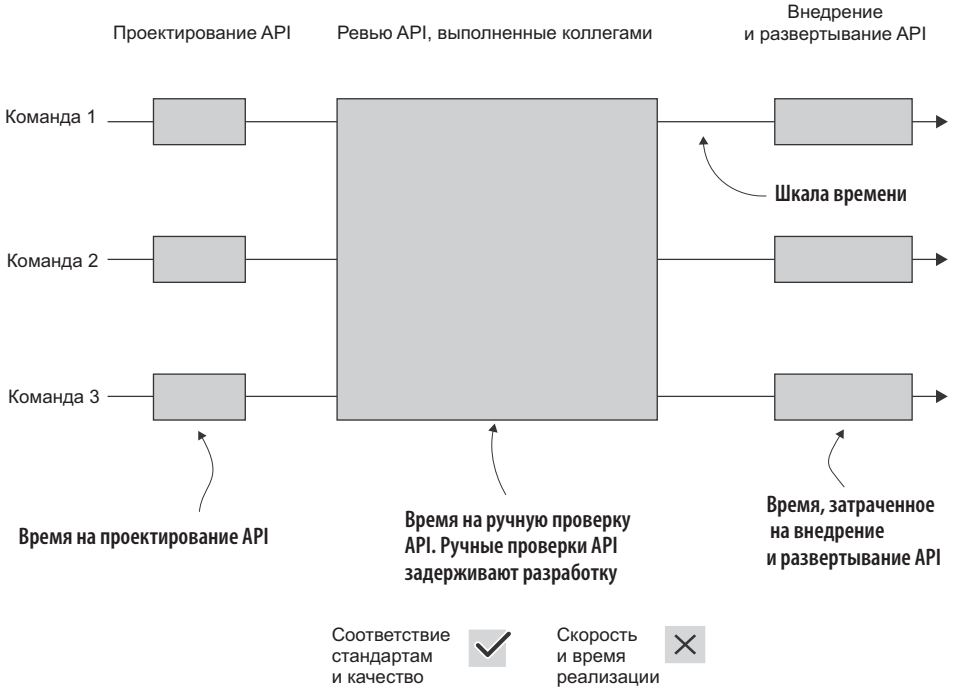


Рис. 1.8. Централизованное ручное управление помогает придерживаться стандартов, но замедляет процесс разработки

APIOps предлагает третий подход — автоматизацию проверок API. Благодаря ей разработка ускоряется, но не в ущерб высоким стандартам качества и соответствию требованиям к API. Обратите внимание, что даже в APIOps могут потребоваться ручные проверки: в одних случаях процессы бывают еще не автоматизированы, а в других — выполняются исключительно человеком. Например, специалисты, ответственные за управление API (API governance), могут вручную проанализировать модель ресурсов, чтобы убедиться, что все участники команды именуют веб-ресурсы и атрибуты единообразно. На этом этапе сохраняется баланс между централизованным и распределенным управлением.

https://liveinternet.club

Автоматизированные проверки соответствия помогают разгрузить команду управления, позволяя ей заняться решением более важных задач, связанных с моделями ресурсов и проектированием операций API. Этап, на котором внедрен подход APIOps, показан на рис. 1.9.

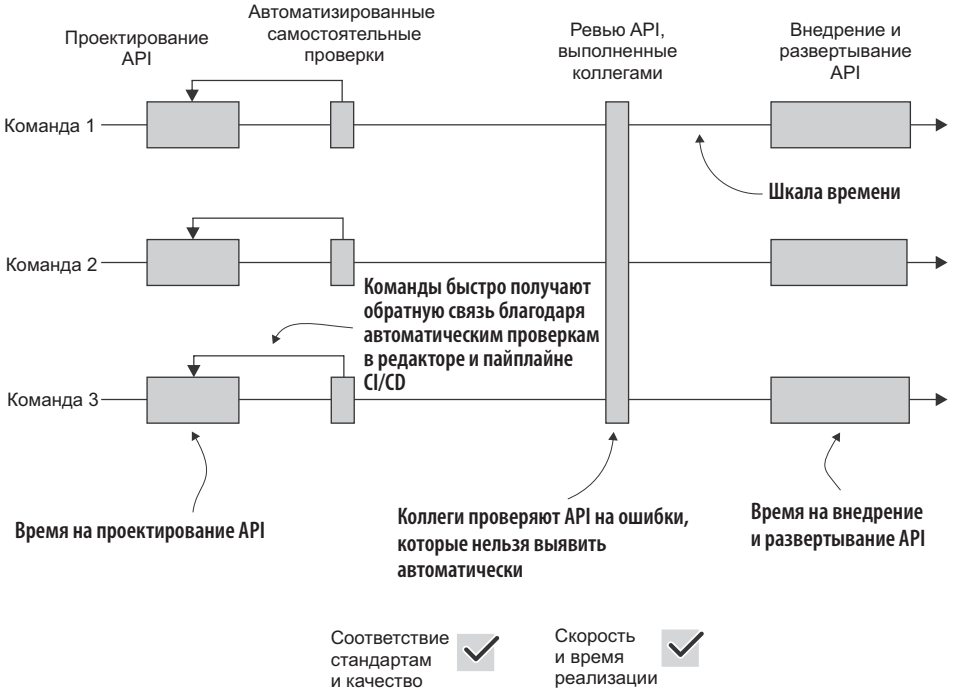


Рис. 1.9. В рамках APIOps управление API максимально автоматизировано, что обеспечивает лучшее соответствие требованиям и сокращение времени разработки

Внедрив APIOps для управления и развертывания API, разработчики получают несколько ключевых преимуществ.

- *Упрощение разработки API*. APIOps предлагает хранить файлы определения и конфигурации API в Git — системе, с которой знакомо большинство разработчиков. Благодаря контролю версий можно автоматизировать процессы пайплайна разработки и упростить создание API.
- *Быстрая обратная связь и снижение стресса*. Так как автоматизированный линтинг выполняется прямо в редакторе для проектирования API и пайплайне CI/CD, разработчики могут быстро получать результаты проверки на соответствие стандартам. APIOps снижает нагрузку на разработчиков, избавляя их от необходимости читать и запоминать все стандарты API в ор-

https://liveinternet.club

ганизации. При этом исключаются трудоемкие ручные проверки, которые все равно чреваты ошибками и могут раздражать из-за неожиданных исправлений на поздних этапах.

- *Улучшенный поиск и повторное использование API.* Хранение файлов описания и конфигурации API в общей системе контроля версий упрощает поиск и повторное использование существующих компонентов и паттернов проектирования API.

Основные преимущества для команд эксплуатации (платформы) API.

- *Быстрое и плавное развертывание.* Благодаря APIOps больше не нужно развертывать изменения вручную. Они проходят через пул-реквесты в репозитории Git и пайплайне CI/CD.
- *Единообразие и соответствие стандартам API.* После внедрения APIOps компаниям больше не нужно выбирать между единообразием и скоростью — то есть централизованными, но полностью ручными проверками (этап 2 выше) и отсутствием стандартов API (этап 1 выше). Автоматизированные проверки и использование Git в пайплайне CI/CD гарантируют, что определения API всегда будут соответствовать установленным требованиям.
- *Сокращение среднего времени восстановления* (Mean Time to Restore, MTTR). Если из-за ошибки в конфигурации API происходит сбой, механизмы непрерывного согласования, реализованные в программных агентах, позволяют без труда откатить изменения и вернуться к предыдущей стабильной версии с помощью команды `git revert`.

ПРИМЕЧАНИЕ В некоторых компаниях есть команды, отвечающие за управление API, поддержку API-шлюза и создание его конфигурации для маршрутизации запросов к нужным микросервисам продуктов API. Последнее означает, что они занимаются непосредственно доставкой продукта и фактически представляют собой компонентные команды, которые сосредоточены на технологической инфраструктуре. Такие команды могут создать узкие места в работе, и вы, возможно, захотите этого избежать. В рамках APIOps роль команды API-шлюза иная. Вместо того чтобы разрабатывать и обновлять конфигурации продукта API в API-шлюзе самостоятельно, она передает эту возможность другим командам разработчиков. Теперь ее задача — просматривать пул-реквесты и консультировать по вопросам проектирования и проверки API (однако она по-прежнему отвечает за поддержку API-шлюза). Таким образом, если раньше команда реализовывала технологический компонент, то сейчас она создает платформу API и помогает другим командам ее использовать.

Основные преимущества для менеджеров по продукту API.

- *Сокращенное время реализации.* APIOps сокращает время, необходимое с момента создания и сохранения API в репозитории Git до момента его развертывания. Это позволяет реже переадресовывать задачи и снижает количество рутинной работы, в результате чего продукт быстрее попадает

https://liveinternet.club

на рынок. Важно отметить, что APIOps также поддерживает параллельную разработку. Для этого одни специалисты создают и утверждают файл определения API и, пока идет его реализация, другие пишут тестовые сценарии или могут начать интеграцию с моделью.

- *API как продукт.* Менеджеры по продукту не просто формируют продукт API, но и могут отслеживать и анализировать изменения в его определении.

1.8. Трудности внедрения APIOps на основе OpenAPI

OpenAPI — самый распространенный стандарт для описания REST API. Однако создать файл определения OpenAPI до фактической разработки сервиса API может быть затруднительно. Возникает вопрос: как именно это сделать? Можно написать определение API на YAML вручную, но это отнимает много времени и не всегда удобно. Поэтому важно использовать хорошие инструменты, например Stoplight или Apicurio, которые предлагают графический интерфейс и упрощают создание файлов определения OpenAPI.

Какой бы инструмент вы ни выбрали, он должен поддерживать интеграцию с Git — основным хранилищем файлов API. Хотя многие разработчики уже знакомы с Git, отметим, что в рамках APIOps каждый, кто хочет изменить файлы определения API, должен понимать, как работает эта система и какие ее процессы приняты в команде.

Как я упоминал, внедрение APIOps требует вложений ресурсов: необходимо автоматизировать поток создания ценности и собрать команду API-платформы — она подготовит инфраструктуру, библиотеки и инструменты, благодаря которым разработчики API смогут действовать самостоятельно. Специалисты, занимающиеся API-шлюзом, должны изменить традиционный подход и адаптироваться к новым требованиям.

Итоги

- Продукт API — пакетное API-решение, управляемое командой API и ориентированное на конкретные задачи потребителя.
- APIOps обеспечивает сквозную автоматизацию жизненного цикла API, опираясь на принципы DevOps и GitOps. Этот подход помогает сократить время реализации API, а также повысить его качество, надежность и соответствие стандартам.
- В APIOps на основе OpenAPI файлы определения и конфигурации хранятся в системе контроля версий, такой как Git, а их проверка и развертывание автоматизированы с помощью специальных рабочих процессов.

https://liveinternet.club

2 Бережливый APIOps: решение проблем и внедрение методологии

В этой главе

- ✓ Метод решения проблем A3 для улучшения APIOps
- ✓ Как визуализировать процесс доставки API и выявлять потери
- ✓ Управление внедрением APIOps

В главе 1 я познакомил вас с подходом APIOps, его принципами и преимуществами. В следующих главах подробнее расскажу о программных инструментах и приемах, которые помогут оптимизировать процесс разработки API. Однако прежде чем переходить к средствам автоматизации, стоит ответить на несколько важных вопросов. Даже если инструменты, о которых говорится в книге, предлагают столько преимуществ, как убедиться, что они действительно улучшат процесс доставки API? Одно дело — внедрить автоматизацию, но как понять, повышает ли она эффективность работы с API на всем жизненном цикле? Главная цель автоматизации в APIOps — быстрая доставка более ценных и качественных API, которые клиентам удобно интегрировать и использовать в работе. Автоматизация должна не только сокращать расходы компании, но и по возможности увеличивать прибыль. Но помимо технических и финансовых сторон важно учитывать корпоративную культуру: как ваши коллеги и специалисты из других команд отнесутся к нововведениям?

https://liveinternet.club

Чтобы ответить на эти вопросы, обратимся к принципам *бережливого мышления* (lean thinking). Это подход, при котором компания постоянно повышает качество услуг для клиентов, одновременно экономя ресурсы и сокращая потери в процессе создания ценности. В этой главе я покажу, как оптимизировать процесс доставки API, объединив принципы бережливого мышления и *метод A3 для решения проблем*. Метод A3, разработанный в компании Toyota, — это структурированный итеративный подход к решению проблем и совершенствованию процессов. Он помогает согласовывать действия с коллегами и эффективно собирать информацию о результатах нововведений. Свое название метод получил от формата бумаги A3 (297 × 420 мм), на которой специалисты Toyota фиксировали ход анализа и предложенные решения. Описывая его, я также научу вас составлять карты процессов доставки API. Мы будем использовать специальные техники и обозначения, принятые для *систематизации потока ценности* (Value Stream Mapping, VSM) — метода, который позволяет визуализировать, как распределены работы и время в процессе производства. Он возник на основе карт потока материалов и информации, которые использовались в компании Toyota.

В заключительной части главы я расскажу о видах потерь, возникающих в процессе доставки API, и о том, как вносить и поддерживать изменения в организации для перехода на APIOps. Основное внимание в главе уделено концептуальным инструментам, которые помогут выявить подходящий тип автоматизации, заручиться поддержкой коллег и постепенно изменять процессы для внедрения нового подхода. Здесь вам не понадобится писать код. Вместо IDE и текстового редактора нужны лишь заточенный карандаш, ластик и несколько листов бумаги. Материал, изложенный в этой главе, будет особенно полезен для менеджеров по продукту API, владельцев продуктов и всех, кто стремится внедрить APIOps в своей организации.

2.1. Проблемы, связанные с качеством API и временем реализации

Представьте, что компания Acme Pet Supplies Inc., в которой вы работаете, оказалась в непростой ситуации. Когда-то специалисты Acme быстро разработали несколько REST API для своей платформы, но теперь клиенты постоянно сообщают об ошибках в документации. Они жалуются на низкое качество справочных материалов (сгенерированных на основе файлов определения OpenAPI) и отмечают, что некоторые REST API спроектированы не единообразно и вызывают путаницу. Чтобы решить проблему, компания сформировала специальную команду API-шлюза, которая отслеживает изменения API и отвечает за доступ к этим интерфейсам. Вскоре специалисты команды обнаружили, что многие API, разработанные в Acme, не соответствуют принятому руководству по стилю. Пытаясь это исправить, они обязали разработчиков придерживаться подхода design-first и отправлять на ручную проверку все изменения в API. Это действительно повысило качество API, но привело к новой проблеме:

https://liveinternet.club

разработчики компании стали жаловаться, что из-за длительных ревью процесс проектирования и доставки API сильно замедлился.

Теперь руководство Acme ищет способ не только повысить качество API, но и сократить время, необходимое на дизайн-ревью. Руководитель назначил вас ответственным за эту инициативу. Как вы будете действовать? Кризис, возникший в компании, открывает прекрасную возможность для внедрения APIOps.

Решение проблем с документацией и проектированием API — отличная отправная точка, и APIOps поможет найти баланс между высоким качеством API и скоростью их доставки. Чтобы начать обсуждение APIOps и приступить к совершенствованию процессов, можно использовать лист бумаги формата A3. Так вам будет проще увидеть проблему, проанализировать ее и разработать эффективное решение.

2.2. Метод A3 для реализации APIOps

Метод A3 — наглядный пошаговый подход к решению проблем, который предполагает постоянный диалог с коллегами. Отчет занимает одну страницу листа A3 и основан на цикле PDSA (plan-do-study-act: планируй, выполняй, изучай, действуй), о котором я расскажу в этой главе. Благодаря этому методу вы сможете кратко излагать суть проблемы и пути ее решения, избавившись от необходимости создавать объемные письменные отчеты или громоздкие презентации в PowerPoint. Универсального шаблона для отчета в формате A3 не существует — структура меняется в зависимости от цели и аудитории. В этой книге мы рассмотрим вариант, который состоит из следующих разделов.

- *Тема* — название улучшения, над которым вы работаете.
- *Предпосылки* — бизнес-контекст и цели организации, которые подчеркивают важность решения проблемы.
- *Текущее состояние и постановка проблемы* — описание проблемы и ее влияние на рабочий процесс.
- *Цель* — желаемый результат.
- *Анализ первопричин* — как были выявлены основные причины проблемы.
- *Контрмеры и целевое состояние* — предложенные действия для изменения ситуации.
- *Подтверждение результата* — данные, подтверждающие эффективность примененных контрмер.
- *Последующие действия* — план стандартизации и распространения успешных контрмер, а также способы передачи полученного опыта другим подразделениям организации.

Перечисленные разделы шаблона проиллюстрированы на рис. 2.1, стрелки указывают порядок заполнения отчета.

https://liveinternet.club

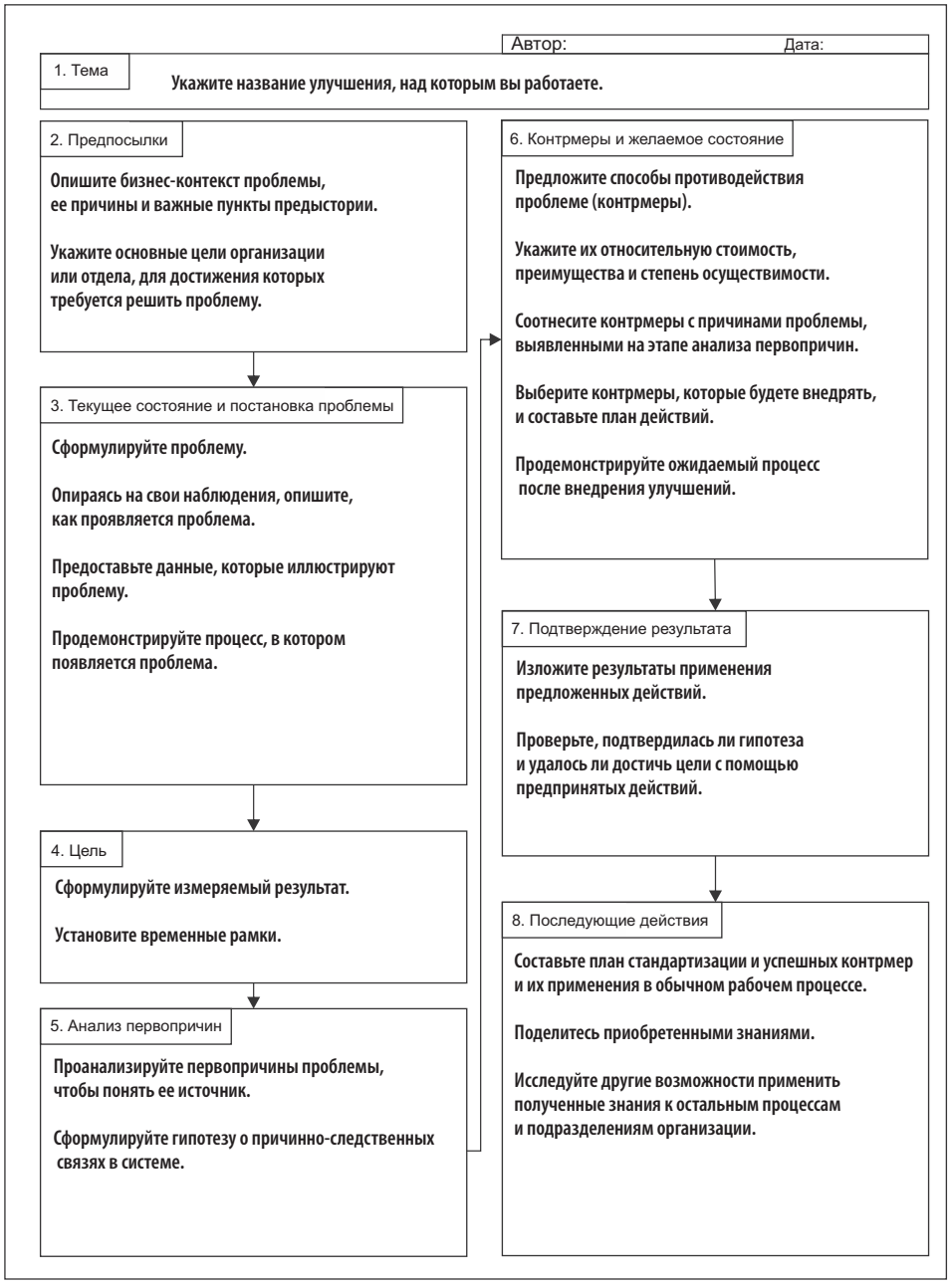


Рис. 2.1. Шаблон отчета в формате A3: разделы и порядок их заполнения

https://liveinternet.club

Вернемся к сценарию из раздела 2.1. Представьте, что вам поручили подготовить отчет в формате A3 о том, как повысить единообразие API, разрабатываемых в компании Acme, и сократить время на дизайн-ревью. С чего начать? Подготовьте необходимые инструменты: карандаш, точилку, ластик и лист бумаги формата A3 — именно они пригодятся вам при заполнении отчета. Если возникло желание обратиться к распространенным цифровым инструментам для построения диаграмм, например Lucidchart, Miro или diagrams.net, — не спешите. Вы рискуете увлечься созданием идеального макета и потратить на это слишком много времени. Заполнение отчета в формате A3 включает несколько циклов размышлений, обсуждений и создания набросков, поэтому вам предстоит неоднократно его менять. Первый набросок лучше сделать карандашом на бумаге, а не в графическом редакторе: это позволит быстрее начать работу и сосредоточиться на решении проблем, обозначенных в отчете, а не тратить время на оформление диаграммы. Когда черновик будет готов для обсуждения с коллегами и руководителем, вы сможете перенести его в цифровой формат.

Виды отчетов в формате A3

Как я уже говорил, все отчеты в формате A3 имеют схожую структуру, но строгого стандарта для их оформления не существует. Главное — возможность структурировать свои мысли с помощью цикла PDCA (plan-do-check-act: планируй, выполняй, проверяй, действуй) и понятно донести их до аудитории. Подробнее метод A3 и его роль в концепции бережливого производства описал Джон Шук (John Shook) в книге «Managing to Learn: Using the A3 Management Process to Solve Problems, Gain Agreement, Mentor, and Lead»[1] (Lean Enterprise Institute, 2008).

Важно отметить, что виды отчетов в формате A3 различаются. Шаблон на рис. 2.1 предназначен для решения проблем, но бывают и варианты, с помощью которых можно готовить идеи и предложения или отчеты о текущем статусе проекта. В зависимости от цели структура отчетов может варьироваться. Более подробную информацию о видах отчетов в формате A3 вы найдете в книге Дурвада Собека и Арта Смолли (Durward Sobek, Art Smalley) «Understanding A3 Thinking: A Critical Component of Toyota's PDCA Management System» (CRC Press, 2008).

2.3. Тема и предпосылки

Тема отчета — это заголовок, который отражает суть предлагаемого улучшения. Он показывает читателю, какое изменение описано в документе и какую проблему вы пытаетесь решить с его помощью. Чтобы начать формирование отчета, запишите тему в верхней части листа A3. В нашем примере она будет звучать так: «Повысить единообразие API и сократить время на дизайн-ревью». Вместе

[1] Дж. Шук. «Метод A3 в управлении компанией: как создать среду развития и сотрудничества».

https://liveinternet.club

с темой читателю важно знать, кто именно отвечает за решение проблемы, включая не только результат, но и действия для его достижения. Поскольку руководитель поручил это задание вам, укажите в отчете свое имя. Помимо темы, в верхней части документа нужно указать дату. В процессе решения проблемы вам предстоит раз за разом возвращаться к отчету по мере получения новой информации. Указывая дату, вы поможете читателю отличать актуальную версию от предыдущих. Запишите тему, свое имя и дату заполнения документа, как показано на рис. 2.2.

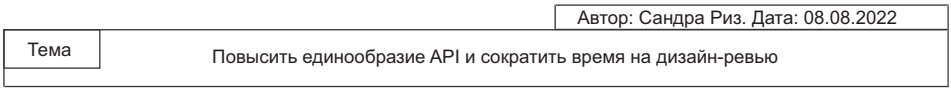


Рис. 2.2. Тема отчета в формате A3 отражает суть предлагаемого улучшения

Формулируя тему, постарайтесь не упоминать причину изменения и готовые решения. Ведь одна из целей составления отчета — исследовать проблему и выявить ее основные источники. На этом этапе заголовок должен сообщить читателю суть предлагаемых нововведений. Обращаясь к методу A3, я использую простое и надежное правило: начинать тему со слов *улучшить*, *повысить*, *сократить*, *упростить* и т. д.

Далее следует раздел «Предпосылки». При знакомстве с ним ваш руководитель должен ясно понять, почему важно решить проблему. Представьте ему бизнес-контекст и свяжите проблему с целями компании или отдела. Для этого полезно ориентироваться на квартальные или годовые планы организации. Здесь же стоит указать предысторию проблемы и подкрепить информацию наглядными материалами: изображениями, таблицами или списками. Пример оформления раздела «Предпосылки» для компании Acme показан на рис. 2.3.

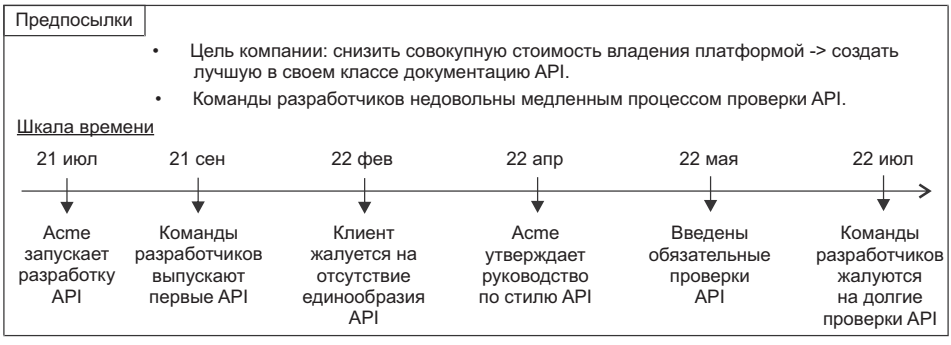


Рис. 2.3. Раздел «Предпосылки» в отчете сообщает читателю контекст проблемы

https://liveinternet.club

Заполняя раздел «Предпосылки», следует ориентироваться на аудиторию. Объясните, почему необходимо внести улучшение, и отметьте, что оно не терпит отлагательств. Списки и графики помогут донести информацию в сжатом виде и привлекут внимание читателя. А изображения зачастую передают гораздо больше, чем слова. В качестве иллюстраций можно использовать гистограммы, круговые диаграммы, линии тренда, шкалы времени и диаграммы Парето.

2.4. Текущее состояние и постановка проблемы

Итак, вы описали руководителю предпосылки проблемы, и теперь пора ее сформулировать. Представим, что в компании Acme каждый запрос клиента на усовершенствование API фиксируется владельцем продукта. Затем инженер дорабатывает API и передает его на ревью и утверждение команде API-шлюза. Ее специалисты собираются дважды в неделю, чтобы рассмотреть API, созданные всеми разработчиками компании. Если обнаруживаются недочеты, инженера просят их исправить, и только после исправления утверждают API. Процесс ревью и доработки оказывается слишком долгим, и разработчикам не удается своевременно выполнять обязательства перед клиентами. Итак, общая картина ясна. Возникает вопрос: как отразить данные и факты в отчете так, чтобы руководитель — а впрочем, и любой читатель — понял суть проблемы и увидел ее проявления?

В разделе «Текущее состояние и постановка проблемы» кратко, в одном-двух предложениях, сформулируйте проблему и проиллюстрируйте, как она проявляется в текущем процессе. Опирайтесь на проверенные факты и точные данные: укажите, как давно она существует и как часто возникает. Чтобы по-настоящему разобраться в проблеме, нужно увидеть, как она показывает себя в реальных условиях. Наблюдение — один из важнейших этапов анализа и решения трудностей. В концепции бережливого производства это называется «выход в *гембу*» — от японского слова «гемба», означающего «настоящее место», где выполняется работа. К примеру, чтобы понаблюдать за ситуацией в компании Acme, можно предпринять следующие шаги.

- Поговорить с коллегами: спросить, какие трудности они испытывают и с какими препятствиями в работе сталкиваются.
- Попросить у коллег разрешения понаблюдать за рабочим процессом, чтобы узнать, как проявляется проблема. Например, если сотрудник работает в офисе, сядьте рядом и посмотрите, как он выполняет задачи, а если удаленно — попросите продемонстрировать или записать изображение на экране.
- Обратить внимание на способ выполнения задачи. Какие применяются инструменты? Какие, наоборот, остаются невостребованными? Например, если в компании утверждено руководство по стилю или рекомендуемый инструмент для проектирования API, посмотрите, используются ли они.

https://liveinternet.club

- Посетить встречи, на которых поднимают и обсуждают проблемы, а также принимают важные решения. В случае с Acme стоит понаблюдать за командой API-шлюза во время проверки API.
- Проанализировать доступные тикеты на исправление ошибок и инцидентов, а также другие документы, в которых может быть зафиксировано проявление проблемы.
- Во время наблюдения воздержитесь от предложений и рекомендаций. Просто сосредоточьтесь на процессе и задавайте вопросы: важно выяснить, почему, когда, кто, что и как. Записывайте всю полученную информацию.

СОВЕТ Лучше всего понаблюдать за процессом три-четыре раза и получить усредненное значение метрик на каждом шаге.

Любой рабочий процесс состоит из нескольких этапов. Внедряя улучшения (в случае с Acme — оптимизируя процесс проектирования и ревью), важно учитывать две оси. Первая отражает общее время процесса, которое складывается из длительности каждого этапа, за исключением параллельно выполняемых задач. Вторая — качество выходных данных, выражаемое через совокупное качество на каждом этапе. Кроме того, при наблюдении за процессом следует фиксировать несколько важных метрик.

- *Время выполнения* (Task Time, TT) — время активной работы над задачей при условии, что у команды есть вся доступная информация, и сотрудников ничто не отвлекает. Сюда также входит время на чтение, анализ, обсуждение и понимание требований. При расчете этой метрики не учитывается время ожидания задачи в бэклоге и другие обстоятельства, из-за которых сотрудники прерывают начатую работу. Время выполнения задачи также называют «*временем обработки*».
- *Время реализации* (Lead Time, LT) — время на одном этапе от постановки задачи до ее выполнения и передачи на следующий этап. Оно включает в себя все задержки, возникающие из-за перерывов в работе, ожидания информации или внешних факторов, препятствующих началу выполнения задачи. Другими словами, время реализации учитывает задержки и время ожидания в очереди (бэклоге). Разница между временем выполнения задачи и временем реализации продемонстрирована на рис. 2.4. Время реализации также называют «временем производства» и «сквозным временем» (throughput time).
- *Процент завершенных и корректных задач* (Percentage Complete and Accurate, % CA) — доля задач, которые поступают от предыдущих этапов и готовы к работе благодаря тому, что не содержат ошибок, не требуют дополнительной информации и не ограничены внешними факторами. Иначе говоря, это процент поступающей работы, за которую можно браться без предварительных исправлений, дополнений и уточнений. Он отражает качество выходных данных на каждом этапе процесса. Для расчета % CA можно спросить коллег на определенном этапе процесса, как часто задачи от предыдущих этапов

https://liveinternet.club

полностью готовы к работе. Например, сотрудники, проверяющие API, могут отметить, что получают всю необходимую информацию для проверки только в 80 % случаев.

- *Незавершенное производство* (Work-in-process inventory, WIP inventory) — количество задач и тикетов, ожидающих обработки. В компании Acme примером незавершенного производства может служить бэклог тикетов на проверку API от разработчиков, который проверяется дважды в неделю. Чтобы выявить незавершенную работу, обратите внимание на этапы, в которых задачи выполняются с определенной периодичностью либо удерживаются, пока не накопится некоторое количество тикетов, а затем обрабатываются разом.

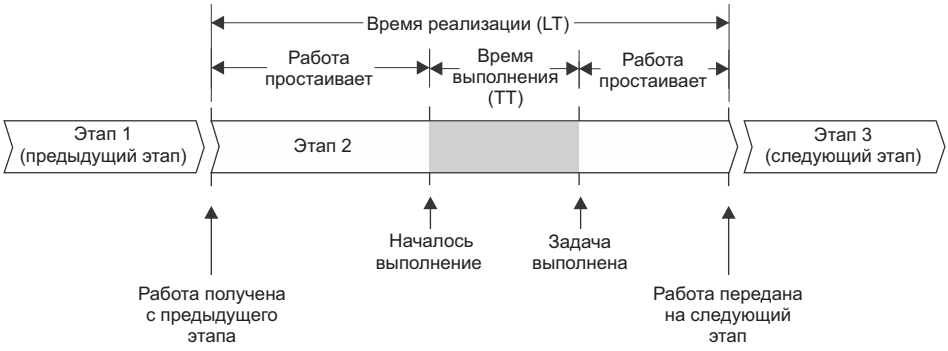


Рис. 2.4. Время выполнения задачи и время реализации на одном из этапов процесса

Помимо перечисленных метрик, можно собрать и другие данные, важные для анализа. Например, сколько тикетов на исправление ошибок связано с исследуемой проблемой, как часто она возникает (в случае с Acme — есть ли в году периоды, когда тикеты рассматриваются особенно долго?), сколько сотрудников задействовано на разных этапах процесса и т. д.

Итак, вы опросили участников процесса, проверили факты и собрали важные данные, демонстрирующие проявление проблемы. Теперь важно изложить информацию в отчете формата А3 так, чтобы руководитель мог быстро вникнуть в суть, не читая пространные описания ваших находок. Как же это сделать?

Опираясь на данные, полученные при «выходе в гембу», представьте проблему визуально. Проиллюстрируйте ее с помощью диаграмм, о которых я говорил выше: гистограмм, круговых диаграмм, диаграмм Парето или других.

СОВЕТ Диаграмма Парето — отличный способ визуализировать данные о частоте возникновения проблемы. Она помогает сосредоточиться на факторах, оказывающих наибольшее влияние на процесс. Обсуждение этого типа гистограмм выходит за рамки моей книги, но вы можете найти дополнительную информацию на странице https://asq.org/quality-resources/pareto.

https://liveinternet.club

Создайте карту процесса, отразив на ней метрики, собранные для каждого этапа. Существуют разные подходы к моделированию бизнес-процессов, но я рекомендую обратиться к техникам и обозначениям, принятым в методе картирования потока создания ценности (Value Stream Mapping, VSM). Как отмечалось выше, он позволяет визуализировать и анализировать ход работы, потоки информации и другие действия, необходимые для выполнения запроса клиента. Обычно с помощью этого метода рассматривают общую картину: он демонстрирует, как повышается ценность продукта или семейства продуктов в ходе основных процессов организации. Тем не менее техники и обозначения VSM можно адаптировать для создания более детальной карты процессов. Подробнее об этом — далее.

2.4.1. Создание этапов рабочего процесса

Для начала на карте нужно отметить клиента, который обратился с запросом, и основные этапы процесса. В компании Acme внутренним клиентом выступает менеджер по продукту, запросивший изменение API. Изобразите клиента в верхней части листа с помощью специального значка (значки, которые я использую, и их описание можно найти в приложении А). Затем обозначьте соответствующими значками основные этапы процесса. Если этапы могут меняться, нарисуйте только те, что встречаются в 80 % случаев, чтобы сосредоточить усилия и не отвлекаться на мелочи. Вверху каждого значка этапа укажите название процесса (желательно в формате «глагол + существительное»[1], например «Спроектировать операции API»). Внизу — команду или сотрудника, ответственного за выполнение задачи. Соедините этапы стрелками так, чтобы обозначить передачу работы между ними. На рис. 2.5 показан пример схемы для компании Acme.

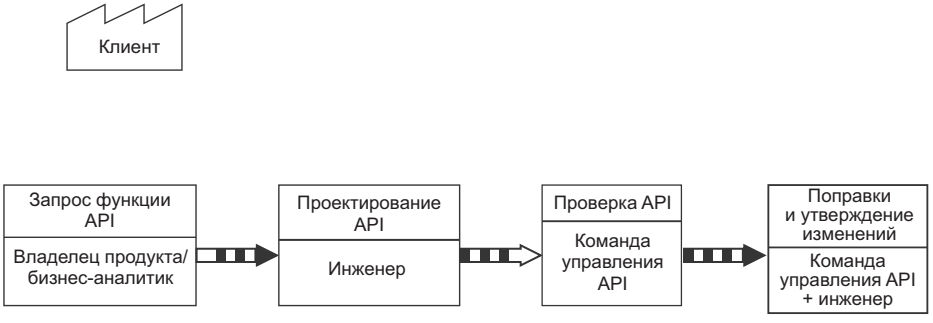


Рис. 2.5. Нарисуйте значок клиента и обозначьте основные этапы процесса

[1] Формат «глагол + существительное» распространен в зарубежной практике, тогда как в российских производственных системах придерживаются варианта «существительное + существительное», например «Проектирование операций API». — *Примеч. пер.*

https://liveinternet.club

2.4.2. Схема потока информации

Определите основные информационные системы для хранения и передачи данных на каждом этапе потока создания ценности. Их анализ поможет обнаружить некоторые виды потерь в ходе производства, о чем мы подробнее поговорим в следующем разделе. Соедините блоки процесса и информационные системы с помощью стрелок. Их направление должно демонстрировать движение данных: стрелка, направленная к системе, означает ввод данных, а отходящая от системы — их извлечение.

Вернемся к компании Acme. Владелец продукта или бизнес-аналитик команды обсуждает задачу с инженерами и фиксирует требования, создавая тикет в Jira. Затем инженеры проектируют API, создавая или обновляя файл определения OpenAPI. При необходимости они сверяются с тикетом в Jira и руководством Acme по стилю, размещенному в Confluence. Для проектирования API используется облачный инструмент, который также служит хранилищем файлов определения OpenAPI. Завершив работу, инженеры создают тикет на проверку API для команды API-шлюза. Когда специалисты команды приступают к задаче, они открывают облачный инструмент для проектирования API, анализируют файлы определения и оставляют комментарии. Линии, обозначающие описанные потоки информации, показаны на рис. 2.6.

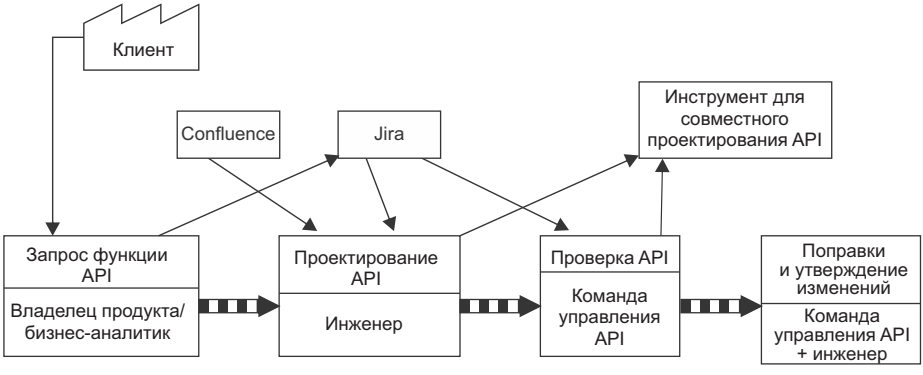


Рис. 2.6. Добавьте информационные потоки

2.4.3. Добавление метрик производительности

На основе своих наблюдений за потоком создания ценности дополните каждый этап метриками производительности, включая время реализации (LT), время выполнения задачи (TT) и процент завершенных и корректных задач (% CA). Поместите их в *блоки данных* — специальные поля для записи важной информации. Кроме того, следует учесть количество задач, ожидающих обработки на каждом этапе. Если визуализировать очереди или бэклог, будет проще

https://liveinternet.club

проанализировать зоны потерь. Обозначьте необработанные задачи значком треугольника («очередь»). К примеру, в компании Acme можно указать количество тикетов в Jira.

Теперь внесите метрики производительности для каждого этапа. Добавьте *лестницу временных интервалов*: сложите метрики времени реализации и времени выполнения задачи для каждого этапа и подсчитайте их итоговое значение. При этом время выполнения задачи укажите сверху шкалы, а время реализации — снизу. Заполненный раздел «Текущее состояние и постановка проблемы» для компании Acme показан на рис. 2.7.

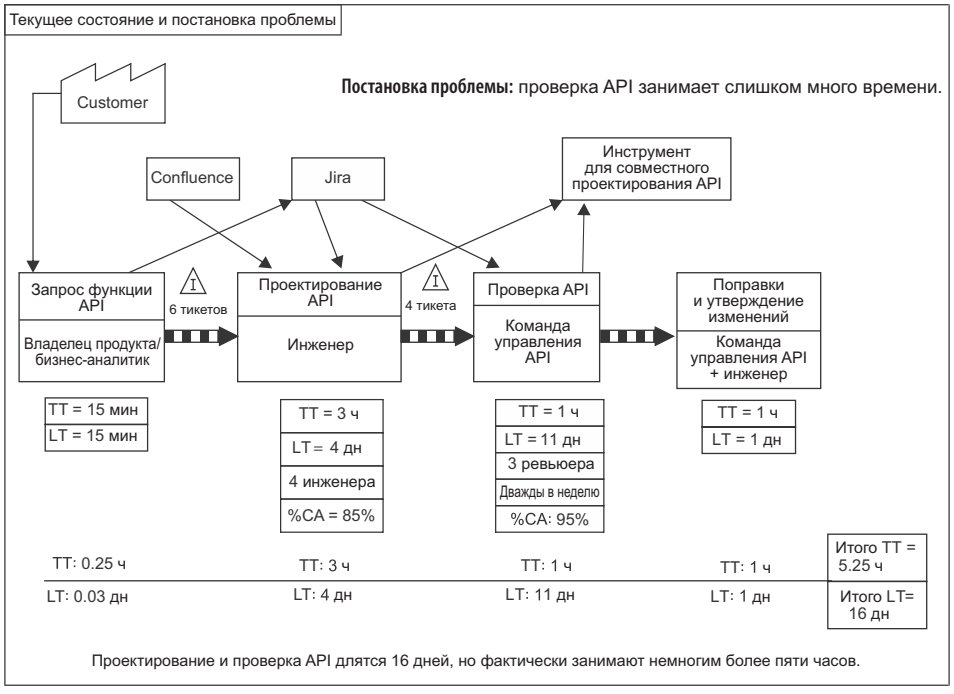


Рис. 2.7. Карта процесса и метрики производительности в разделе «Текущее состояние и постановка проблемы»

ПРИМЕЧАНИЕ В этой книге я рассматриваю карты по методу VSM только для оптимизации APIOps. Если вас интересует систематизация потоков ценности в других сферах, обратитесь к книге Майка Ротера и Джона Шука (Mike Rother, John Shook) «Learning to See»[1] (Lean Enterprise Institute, 2009) или книге Карен Мартин и Майка Остерлинга (Karen Martin, Mike Osterling) «Value Stream Mapping» (McGraw-Hill,

[1] М. Ротер, Дж. Шук. «Учитесь видеть бизнес-процессы. Построение карт потоков создания ценности».

https://liveinternet.club

2013). Кроме того, рекомендую изучить *создание карт на основе метрик* (Metric-Based Process Mapping, MBPM) — подход для подробного описания процессов, в котором метрики определяются совместно с коллегами при помощи магнитной доски, листа бумаги и стикеров. Подробнее см. в тематической статье (https://mng.bz/y8NG) и на слайдах Карен Мартин (https://mng.bz/OZWR).

2.4.4. Анализ карты

Посмотрим, какие проблемы выявляет карта процесса на текущем этапе. Из рис. 2.7 видно, что владелец продукта или бизнес-аналитик Acme создает тикет на изменение API за 15 минут (0,25 часа). Время выполнения задачи совпадает со временем реализации, поэтому задержек не возникает. Обновление API (файла определения) занимает у инженеров три часа. В команде работают четыре инженера, но прежде чем приступить к этому тикету, им нужно выполнить еще шесть задач. В результате тикет на изменение API проведет несколько дней в бэклоге. Предположим, время реализации на этом этапе составляет четыре дня. Понаблюдав за рабочим процессом и поговорив с инженерами, вы выяснили, что они получают всю необходимую информацию (из тикета Jira и руководства по стилю API в Confluence) примерно в 85 % случаев. Для выполнения оставшихся задач им не хватает данных, а значит, приходится запрашивать их и ждать разъяснений. Таким образом, здесь % CA составляет 85 %.

Доработав API, инженеры отправляют команде шлюза тикет в Jira на проверку. Эта команда — центральное звено, которое анализирует все изменения в API и следит за соблюдением руководства по стилю API. Однако к ней обращаются сразу 16 команд разработчиков Acme. Сейчас в бэклоге ожидают четыре тикета на ревью изменений в API, а также множество других запросов, связанных с маршрутизацией и тестированием шлюза API. Чтобы распределить нагрузку, команда проверяет и утверждает изменения дважды в неделю — по вторникам и четвергам, при этом обсуждение каждого тикета длится около часа. Учитывая четыре тикета в бэклоге, можно сделать вывод, что команда проверит наши изменения через 11 рабочих дней. При проведении ревью команда API-шлюза приглашает инженеров на совещание и просит объяснить изменения. Отчасти это связано с тем, что инструмент Acme для проектирования API не поддерживает функцию diff, которая показывала бы различия между версиями, и команде сложно понять, что именно изменилось в отправленном файле определения API.

Во время совещания специалисты команды выявляют три ошибки и поручают инженерам их исправить. Наконец, получив исправленную версию, команда API-шлюза окончательно утверждает API. Внесение и утверждение правок занимает всего час, но инженеры выполняют и другие задачи, поэтому некоторое время уходит на ожидание. В итоге время реализации этого этапа (LT) составляет один день (8 рабочих часов).

На лестнице временных интервалов показано, что общее время реализации (LT) достигает 16 рабочих дней (128 часов) — довольно много для утверждения всего

https://liveinternet.club

одной операции API. Однако общее время фактического выполнения задачи (TT) составит всего 5,25 часа. Значит, *коэффициент активности* (activity ratio), равный отношению суммарного TT к суммарному LT, равен 4,1 %. Оставшиеся 95,9 % времени задача простаивает в очереди и никак не продвигается в потоке создания ценности. Не очень-то эффективно! Как видите, карта процесса помогла нам обнаружить потери.

2.5. Цель

Итак, вы проиллюстрировали текущие показатели процесса, и ваш руководитель теперь видит, как проявляется проблема. Однако прежде чем рассматривать потенциальные улучшения, он хочет узнать: по каким критериям будет измеряться успех? Как понять, что процесс действительно стал лучше? Ответить на эти вопросы и заранее продумать, какие показатели подтвердят эффективность изменений, поможет заполнение раздела «Цель».

Опираясь на анализ текущего состояния, выберите одну или две метрики производительности, которые вы хотите улучшить. К примеру, в компании Acme можно уменьшить суммарное время утверждения API на 20 %. То есть общее время реализации процесса сократится с 16 дней до 12,8 дня. Кроме того, можно указать срок достижения цели: от пары недель до нескольких кварталов. Так как при составлении отчета в формате A3 рекомендуется представлять информацию в виде графиков, диаграмм и таблиц, предлагаем проиллюстрировать цель с помощью простой таблицы, как показано на рис. 2.8.

Цель

Сократить время реализации на 20 %
Длительность улучшения: 4 спринта (спринт 153 — спринт 156)

Метрика	Текущее состояние	Желаемое состояние	Прогнозируемое улучшение, %
Время реализации	16	12.8	20%

Рис. 2.8. Пример цели с описанием в виде таблицы

Срок достижения цели можно задать сразу или позже, после заполнения раздела «Контрмеры». Я работаю с командами по методологии Agile и привык выражать сроки в количестве спринтов, необходимых для достижения результата.

https://liveinternet.club

Отмечу, что раздел «Цель» не предназначен для обсуждения решений. Здесь нужно зафиксировать измеримую цель — метрики производительности, которые позволят объективно оценить успех. После внедрения улучшений вы сможете вернуться к этим метрикам и проверить, удалось ли их достичь.

2.6. Анализ первопричин

К текущему этапу вы уже определили проблему, выявили ее проявления и поставили цель для улучшения процесса. Но чтобы действительно решить проблему, необходимо понять, почему же она возникла. Найти и устранить ее источник и тем самым предотвратить повторение ошибок поможет анализ первопричин. Одна из наиболее распространенных техник называется «Пять почему». Используя ее, необходимо несколько раз спросить себя: «Почему?» или «Что вызвало эту проблему?» Важно понимать: количество вопросов варьируется и бывает больше или меньше пяти, а одна проблема может иметь несколько причин возникновения. Ваша цель — выявить среди них истинную, чтобы предотвратить повторение ошибок. Дерево анализа первопричин для компании Acme показано на рис. 2.9.

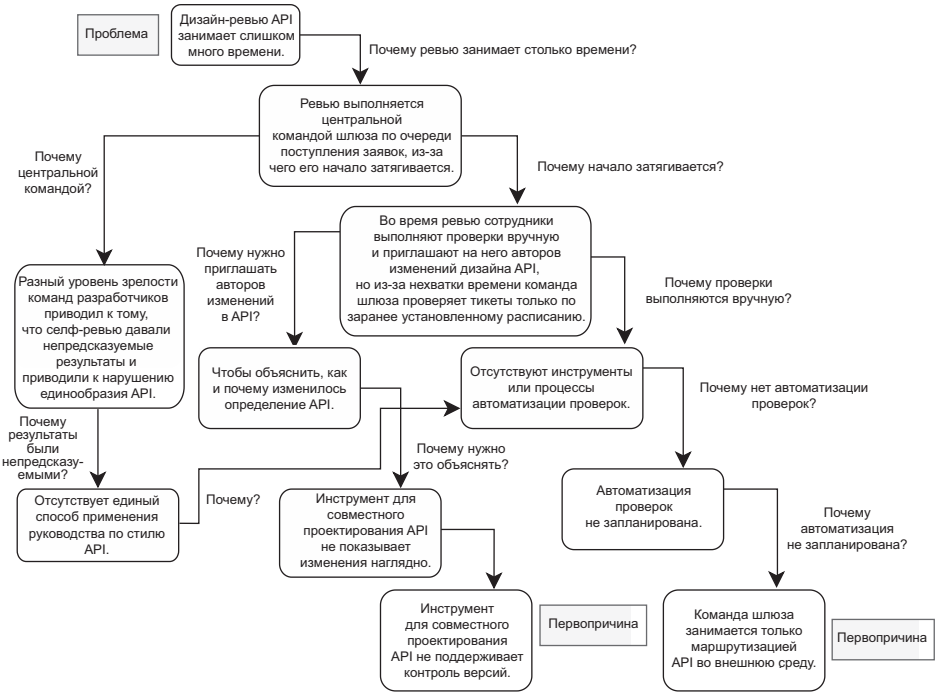


Рис. 2.9. Анализ первопричин, по которым в компании Acme затянулось время реализации изменений в API

https://liveinternet.club

Анализ позволил выявить две первопричины. Во-первых, используемый инструмент для совместного проектирования API не поддерживает контроль версий. Во-вторых, команда API-шлюза, центральное звено, — компонентная. Она занимается маршрутизацией API и не автоматизирует проверки и другие этапы сквозного проектирования и доставки API.

Анализ первопричин помогает найти и устранить глубоко укоренившиеся источники проблемы и тем самым исключить возможность ее повторного возникновения. В его основе лежит логическое и критическое мышление, которое позволяет выявить причинно-следственные связи. Благодаря ему вы избежите поспешных выводов и полумер, которые лишь временно маскируют проблему. Если вместо поверхностного изучения симптомов и мнений копнуть глубже, можно обнаружить истинную причину. В разделе «Анализ первопричин» вы описываете гипотезу о том, как работает ваша система. Чаще всего гипотеза сводится к тому, что устранение первопричин позволит улучшить всю систему. Во время анализа обсудите свои мысли с коллегами и другими стейкхолдерами: они могут подтвердить ваши выводы или назвать причины, о которых вы не задумывались.

СОВЕТ Помимо подхода «Пять почему», можно использовать еще один инструмент мышления — диаграмму Исикавы (также известную как «рыбья кость»). Ее обсуждение выходит за рамки моей книги, но вы можете узнать о ней больше на странице https://asq.org/quality-resources/fishbone.

2.7. Контрмеры и целевое состояние

Заполняя предыдущие разделы отчета в формате A3, вы исследовали саму проблему: сформулировали ее, изучили предпосылки, изложили данные и проанализировали первопричины. Теперь пора задуматься о решении. В словаре бережливого производства действия, направленные на устранение первопричин проблемы, называются *контрмерами*. В контексте APIOps контрмеры представляют собой приемы и методы, которые, по вашему мнению, помогут улучшить процесс доставки API.

Чтобы заполнить раздел «Контрмеры и целевое состояние», проведите мозговой штурм. Запишите в таблицу все возможные действия для устранения первопричин, выявленных во время анализа. Чтобы найти новый, нестандартный подход, рассмотрите все варианты, которые придут вам в голову, и обсудите их с другими участниками процесса. Привлекая специалистов из разных отделов и учитывая общие интересы, можно подобрать оптимальные контрмеры. Постепенно сократите список идей, объединив похожие действия и исключив наименее осуществимые.

Затем подготовьте план реализации. Для этого укажите в таблице, кто будет отвечать за контрмеры и к какому сроку необходимо их реализовать. Чтобы

https://liveinternet.club

расставить приоритеты, добавьте столбец «Ожидаемая польза/сложность», где польза принимает значения «высокая/средняя/низкая», а сложность — «низкая/средняя/высокая». Наиболее приоритетными должны быть контрмеры, которые приносят максимальную пользу при минимальной сложности. Менее полезные и более трудновыполнимые действия можно отложить или исключить. По мере реализации контрмер обновляйте метку в столбце «Статус». Пример оформления таблицы показан на рис. 2.10.

Причина	Контрмеры	Ответственное лицо	Ожидаемая польза/сложность	Срок	Статус
Команда шлюза занимается только маршрутизацией API во внешнюю среду	Команда шлюза должна запланировать внедрение автоматизации проверок	Команда шлюза	Высокая/низкая	2023/01/06	Done
Команда шлюза занимается только маршрутизацией API во внешнюю среду	Создать систему автоматизированного линтинга и выявления критических изменений	Команда шлюза	Высокая/средняя	2023/02/24	Done
Инструмент для совместного проектирования API не поддерживает контроль версий	Перенести определения API в Git	Команда шлюза	Высокая/средняя	2023/01/20	Done
Инструмент для совместного проектирования API не поддерживает контроль версий	Проводить проверки API через пул-реквесты с привязкой к истории пользователя	Команда шлюза	Высокая/низкая	2023/01/23	Done

Рис. 2.10. Контрмеры — это действия, предпринимаемые для устранения первопричины проблемы. Таблица контрмер помогает составить план и определить, кто, что и когда должен сделать

> **СОВЕТ** Не пытайтесь охватить сразу всю проблему, сосредоточьтесь на некоторых ее аспектах. На этом этапе вы экспериментируете и проверяете, какие контрмеры работают, а какие нет. Например, в компании Acme можно сначала протестировать изменения на одном API или в одной команде разработчиков. Если результат окажется положительным, на следующем этапе можно масштабировать контрмеры на весь процесс.

В концепции бережливого производства понятие *контрмера* используется вместо слова *решение*, поскольку контрмера устраняет проблему в текущих обстоятельствах, но окончательным решением не является. В будущем может появиться более эффективный вариант или проблема изменится и потребует иных действий. Поскольку ключевой принцип бережливого производства — непрерывное улучшение процессов, необходимо регулярно анализировать возникающие проблемы и совершенствовать контрмеры. Эти действия направлены на оптимизацию *потока*. Работа должна переходить от одного этапа к другому без задержек, прерываний, простоев и возвращения на доработку. Внедрение

https://liveinternet.club

APIOps поможет сократить время ожидания и тем самым оптимизировать рабочий поток процесса.

Итак, вы определили контрмеры, но вашему руководителю и другим, кто пожелает ознакомиться с отчетом, может быть непонятен усовершенствованный процесс после их внедрения. Как же пояснить его?

Опишите *целевое состояние* в виде карты измененного процесса. Так руководитель с первого взгляда поймет, какие улучшения произойдут после внедрения контрмер, и увидит ожидаемые метрики производительности на каждом этапе. На рис. 2.11 показан заполненный раздел «Контрмеры и целевое состояние» с картой измененного процесса для компании Acme. Из него видно, что инженеры смогут запускать автоматизированные проверки уже на раннем этапе работы («Проектирование и автоматизированные проверки»). Благодаря возможности проверить API без обращения к команде шлюза сократится время обнаружения ошибок. Также примечательно, что в процессе появится новый этап — автоматизированные проверки в пул-реквесте. Очередь тикетов на ревью исчезнет, а время реализации сократится с 16 дней до 4,1 дня.

Таким образом, для заполнения раздела «Контрмеры и целевое состояние» необходимо:

- провести мозговой штурм, разработать контрмеры и обсудить их с коллегами;
- составить план реализации контрмер — определить ответственных и указать сроки;
- создать карту целевого состояния, на которой наглядно показан оптимизированный процесс.

2.8. Подтверждение результата

Внедрив контрмеры, важно убедиться, что они действительно помогли уменьшить или вовсе устранить проблему. Как это проверить? В разделе «Подтверждение результата» соберите данные, которые покажут, что ваши действия привели к ожидаемым изменениям, установленным в разделе «Цель». Запишите и обобщите полученные знания о внедрении улучшений. Затем выведите четкую причинно-следственную связь между примененными контрмерами и наблюдаемым эффектом. В случае с Acme можно построить диаграмму, которая покажет, как менялось время реализации и соблюдалось руководство по стилю в течение нескольких недель после внедрения контрмер. Отметьте, когда были внесены изменения в процесс. Если гипотеза о причинах проблемы верна, после этих отметок показатели должны улучшаться. Заполненный раздел «Подтверждение результата» для Acme изображен на рис. 2.12.

https://liveinternet.club

Контрмеры и целевое состояние

Причина	Контрмеры	Ответственное лицо	Ожидаемая польза/сложность	Срок	Статус
Команда шлюза сосредоточена на маршрутизации API	Команда шлюза должна запланировать внедрение автоматизации проверок	Команда шлюза	Высокая/низкая	2023/01/06	Готово
Отсутствуют автоматизированные проверки	Создать систему автоматизированного линтинга и выявления критических изменений	Команда шлюза	Высокая/средняя	2023/02/24	Готово
Сложно выявлять различия между версиями во время проверки	Перенести определения API в Git	Команда шлюза	Высокая/средняя	2023/01/20	Готово
Сложно выявлять различия между версиями во время проверки	Проводить проверки API через пул-реквесты с привязкой к истории пользователя	Команда шлюза	Высокая/низкая	2023/01/23	Готово

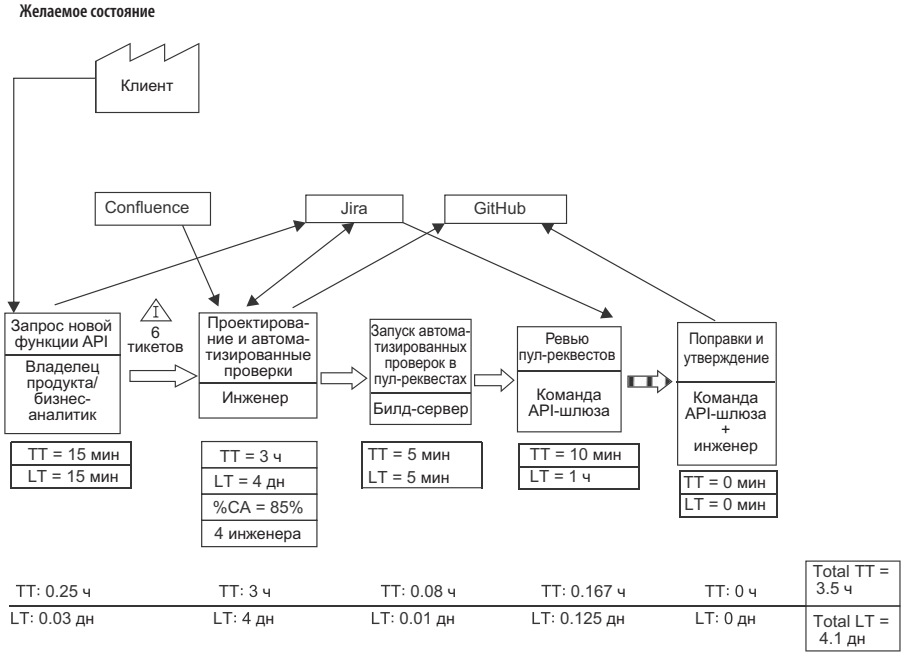


Рис. 2.11. Карта будущего потока ценности

Раздел «Подтверждение результата» помогает проверить, правильно ли вы сформировали гипотезу о причинах проблемы в разделе «Анализ первопричин», и демонстрирует ваше понимание процесса или системы. Подтвердив свои знания о том, как работает процесс, вы заложите фундамент для масштабирования успешных решений на другие процессы или области деятельности организации.

https://liveinternet.club

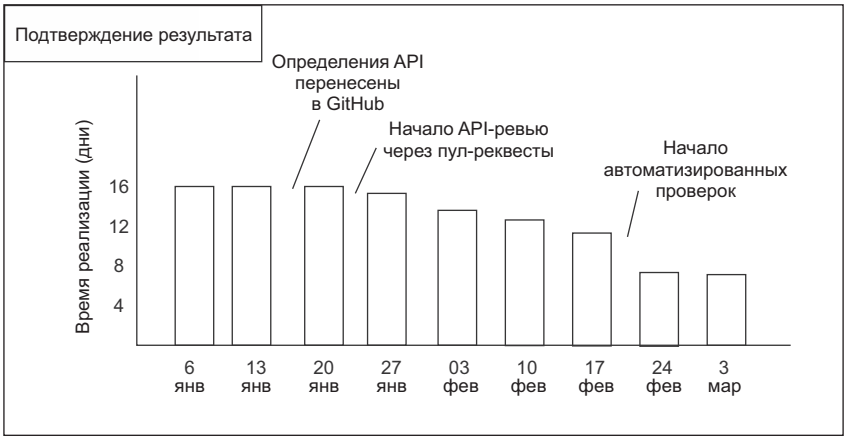


Рис. 2.12. Пример раздела «Подтверждение результата»

2.9. Последующие действия

Предположим, вы внедрили контрмеры в команде разработчиков, которая согласилась поучаствовать в эксперименте, и данные в разделе «Подтверждение результата» показали эффективность нововведений. Как масштабировать успешные решения? К примеру, в компании Acme их можно внедрить в процессы других разработчиков и специалистов API-шлюзов. Заполняя раздел «Последующие действия», вы продумываете распространение контрмер, подтвердивших свою эффективность: планируете истории и эпики, а также определяете, с какими подразделениями организации стоит поделиться выводами. На рис. 2.13 приведен пример таблицы последующих действий для компании Acme.

Последующие действия

Задача	Владелец	Срок	Статус
Применять автоматизированные проверки ко всем типам изменений в дизайне веб-API	Команда шлюза	2023/04/07	В работе
Передать команде инженеров ответственность за ревью пул-реквестов		2023/04/21	Запланировано
Применять автоматизированные проверки к определениям асинхронных API	Команды разработчиков	2023/05/01	Запланировано

Рис. 2.13. Пример раздела «Последующие действия» для компании Acme

https://liveinternet.club

Отчет в формате A3 удобно использовать для обмена опытом. Увидев отчет, подготовленный специалист быстро поймет суть проделанной работы и полученные результаты.

2.10. Ревью отчета в формате A3

Время от времени показывайте отчет руководителю, чтобы убедиться в правильности своих рассуждений. Обратная связь от человека, который знаком с другими процессами организации, поможет оценить точность и объективность вашего решения, разработанного по принципам PDSA. К тому же это отличная возможность обсудить проблему с руководителем, продемонстрировать объем выполненной работы и убедиться, что ваши взгляды на проблему совпадают.

Также рекомендую регулярно просматривать отчет вместе с командой и ключевыми стейкхолдерами. Мне нравится обсуждать решения на встречах профессионального сообщества организации — в некоторых компаниях его называют гильдией API (API Guild). Это хорошая возможность услышать мнение других специалистов, которым небезразлична оптимизация процесса доставки API. Они могут предложить эффективные контрмеры и последующие действия, которых вы, возможно, не учли. Проводите официальные ревью и неформальные обсуждения отчета на всех этапах его подготовки, а также советуйтесь и взаимодействуйте с коллегами, чтобы интегрировать в отчет всестороннюю обратную связь.

2.11. Выявление потерь в потоке ценности

Если вы хотите следовать принципам бережливого производства, важно понимать, какой ценностью обладает продукт. При анализе карты текущего состояния процессов следует отличать действия, которые на самом деле повышают ценность, от тех, которые этого не делают.

Бережливое мышление

Термин *бережливый* (lean) впервые появился в статье Джона Крафчика (John Krafcik) «Triumph of the Lean Production System», опубликованной в журнале Sloan Management Review в 1988 году. Она описывала выдающиеся показатели качества и продуктивности, которых добилась компания Toyota по сравнению с другими автопроизводителями. Д. Крафчик отметил, что философия Toyota основана на постоянном стремлении повышать эффективность и качество продукции при одновременном снижении затрат на ее производство. Значительный вклад в распространение этих идей внесли Джеймс Вумек, Дэниел Джонс и Дэниел Рус (James Womack, Daniel Jones, Daniel Roos), рассказав о подходе в книге «The Machine That Changed the World»[1] (Free Press, 2007). Опираясь на опыт компаний, внедривших эти идеи, Дж. Вумек и Д. Джонс

[1] Дж. Вумек, Д. Джонс, Д. Рус. «Машина, которая изменила мир».

https://liveinternet.club

также написали книгу «Lean Thinking»[1] (Free Press, 2010), где сформулировали пять принципов бережливого производства.

1. Определите ценность каждого продукта.
2. Определите поток создания ценности для каждого продукта.
3. Организуйте бесперебойное движение потока ценности. Откажитесь от больших партий работы и очередей, которые неэффективны по своей природе.
4. Позвольте клиенту вытягивать ценность у производителя.
5. Постоянно стремитесь к совершенству.

Бережливое мышление помогает четко определять ценность продукта или услуги, выстраивать эффективную последовательность действий для ее создания, выполнять задачи с минимальными потерями и по требованию клиента, а также непрерывно совершенствовать процессы. С этим подходом вы научитесь выявлять и устранять различные виды потерь в процессе производства. В результате создание большей ценности для клиента потребует от компании меньше усилий и средств. Одна из целей этой книги — показать вам, как применять принципы бережливого производства для проектирования, разработки, развертывания и публикации API, а также для наиболее эффективного управления этим процессом.

Разработчики, знакомые с методологией Agile, могут вспомнить привычные им фреймворки, например Scaled Agile Framework (SAFe) или Large Scale Scrum (LeSS). Хотя при внедрении Agile в процесс разработки ПО действительно применяются некоторые принципы и методы бережливого производства, важно понимать, что фреймворки Agile от них отличаются.

Всю работу, выполняемую в процессе производства, можно разделить на три категории. Первая включает значимую работу, то есть операции, добавляющие ценность. Ценностью называется то, что потребители считают полезным и важным, а в случае с внешними клиентами — то, за что они готовы платить. Работа считается значимой, если при ее выполнении исходные данные преобразуются в ценный для клиента результат. Например, анализ требований, проектирование решений, написание кода и создание документации API.

Во вторую категорию выделена незначимая работа — операции, которые, хотя и не превращают исходные данные в продукт или услугу для клиента, все-таки необходимы для выполнения значимой работы. Так, приемочное тестирование и проверка безопасности в тестовой среде не улучшают программные продукты напрямую, но помогают выявить проблемы, которые необходимо устранить, прежде чем доставить продукт потребителю. В вымышленной компании Acme проверки API сами по себе не добавляют ценности, но косвенно ее обеспечивают. То же самое обстоит с тикетами, которые создают бизнес-аналитики или владельцы продуктов. Они не добавляют ценности напрямую, однако содержат информацию, необходимую инженерам для проектирования API.

[1] Дж. Вумек, Д. Джонс. «Бережливое производство».

https://liveinternet.club

Действия из третьей категории считаются потерями — работой, потребляющей ресурсы, но не создающей результата, за который клиент готов платить. К примеру, потери возникают из-за простоя, когда вы не можете приступить к работе, ожидая, пока коллега выполнит свою задачу. Подумайте: есть ли в потоке ценности задачи, которые излишне переадресовываются? Или этапы, которые пора объединить? В компании Acme на ожидание тратится 11 дней — вот настоящий пример пустой траты ресурсов!

Чтобы улучшить процесс доставки ценности, необходимо выполнять как можно больше значимых действий, сократить количество незначимой работы и полностью устранить потери.

Семь видов потерь в бережливом производстве

Автор книги «Toyota Production System»[1] (Productivity Press, 1988) Тайити Óно (Taiichi Ohno) выделил семь видов потерь в процессе производства. Он писал: «Мы анализируем временной промежуток с момента получения заказа до его оплаты клиентом. И стараемся сократить это время, избегая действий, которые не увеличивают ценность». Хотя в его книге виды потерь определены для автомобильного производства Toyota, предложенные принципы применимы и в программной разработке, в том числе при создании API. Итак, Т. Оно выделил следующие виды потерь.

- *Производство бракованной продукции*. При создании продуктов API брак может принимать различные формы, включая некорректную справочную документацию и неправильные конфигурации API. Такие проблемы не только приводят к потерям из-за самих ошибок, но и требуют дополнительных ресурсов на поиск и устранение этих ошибок. При этом QA-тестирование не относится к категории значимой работы.
- *Перепроизводство невостребованной продукции*. Проектирование и разработка API или функций API, которые не нужны пользователям (то есть отсутствует экономическая потребность). Помните, что каждая функция API требует времени на разработку, усложняет архитектуру и приводит к альтернативным издержкам (издержкам упущенной выгоды): время, затраченное на ее создание, могло быть использовано эффективнее.
- *Незавершенное производство*. Незавершенные процессы проектирования и доставки API, которые ждут дальнейшей обработки. Характерно для пакетной обработки заявок. Поддержание большого количества незавершенных задач требует затрат капитала и других ресурсов.
- *Избыточная обработка*. В процессе создания API потери могут возникать, например, если команда повторно проектирует компонент OpenAPI вместо применения существующего решения (схемы OpenAPI в общем репозитории) или неоднократно реализует часто используемый код на этапе разработки сервиса API. Нет смысла заново писать код для проверки идемпотентности, если уже существует готовая библиотека.

[1] Т. Оно. «Производственная система Тойоты».

https://liveinternet.club

- *Необязательные передвижения*. На производстве этот вид потерь возникает, если для сборки изделий рабочий вынужден ходить за деталями в другое помещение. Сборка изделий — значимая задача, при выполнении которой материалы преобразуются в готовый продукт. Однако излишнее перемещение в поисках необходимых деталей отнимает время, которое можно потратить на саму задачу. Хотя в этом случае перемещение поддерживает значимую операцию, оно не добавляет ценности напрямую и его можно сократить. Аналогичная ситуация возникает при разработке API: если инженер тратит время на поиск людей, которые объяснят требования к API или откроют доступ к файлу определения, он выполняет незначимую задачу, которая сопровождается потерями.
- *Необязательная транспортировка материалов и готовой продукции*. В контексте разработки API означает ненужное перемещение информации: переадресацию задач, отправку электронных писем, которые не нужны получателям, или бесцельное создание внутренних отчетов и документации.
- *Ожидание необходимого результата работы с предшествующего этапа*. В программной разработке потери возникают, когда сотрудники ждут утверждения правок (например, при ручном ревью API), завершения тестирования и т. д.

Другие примеры, иллюстрирующие семь видов потерь в программной разработке, можно найти в книге Мэри и Тома Поппендик (Mary Poppendieck, Tom Poppendieck) «Implementing Lean Software: From Concept to Cash»[1] (Addison-Wesley, 2007), а также в книге Бодана Оппенхайма (Bohdan Oppenheim) «Lean for Systems Engineering with Lean Enablers for Systems Engineering» (Wiley, 2011). Кроме того, вам может быть интересна электронная книга Марьюкки Ниниоя (Marjukka Niinioja) «The 8 Wastes of Lean in API Development» (Osaango Ltd; https://mng.bz/MZw7), в которой предлагается иная точка зрения на потери в контексте проектирования и доставки API.

Книга, которую вы сейчас читаете, посвящена автоматизации доставки API, но важно отметить, что автоматизация сама по себе не гарантирует роста производительности, если не устранить потери. Когда в вымышленной компании Acme внедрили линтинг, производительность повысилась, однако по карте целевого состояния видно, что этап «Проектирование и автоматизированные проверки» по-прежнему требует внимания из-за высокого времени реализации (4 дня) по сравнению с низким временем выполнения задачи (3 часа). Значит, основное время работа простаивает. Для оптимизации процесса нужно внимательно понаблюдать за действиями на этом этапе, совершив «выход в гембу», — понять, почему возникает простой, выявить потери и определить их первопричины. Это послужит отправной точкой для следующей итерации улучшений по методу A3. Может оказаться, что контрмеры, которые вы предложите на основе наблюдений, вовсе не потребуют автоматизации процессов. Например, если для решения проблемы нужно изменить структуру команды, подход к работе

[1] М. Поппендик, Т. Поппендик. «Бережливое производство программного обеспечения: от идеи до прибыли». — *Примеч. пер.*

https://liveinternet.club

или другие организационные аспекты. Поэтому следует рассматривать как можно больше вариантов. Ваша цель — не автоматизация ради автоматизации, а постоянное повышение метрик производительности (времени реализации, процента выполнения и точности и т. д.) и в итоге максимизация ценности для клиента. Именно этот принцип — стремление к совершенству — лежит в основе бережливого производства.

2.12. Преимущества использования метода A3

При внедрении APIOps мы стремимся добиться сквозной автоматизации проектирования и доставки ценности API. Однако автоматизация имеет смысл только в том случае, если помогает снизить затраты или создать дополнительную ценность для клиентов, например, освобождая ресурсы для более значимых задач. Совершенствование процесса по методу A3 позволяет глубже понять, какие преимущества для организации принесут нововведения. Рассмотрим основные достоинства внедрения APIOps согласно этому подходу.

2.12.1. Развитие логического и объективного мышления

Философия метода A3 основана на упомянутом цикле PDSA (plan-do-study-act: планируй, выполняй, изучай, действуй; иначе его нызывают «цикл Деминга»), предложенном У. Эдвардсом Демингом (W. Edwards Deming), а также на тесно связанном с ним цикле PDCA (plan-do-check-act: планируй, выполняй, проверяй, действуй). Оба подхода помогают управлять изменениями и обеспечивают непрерывное совершенствование процессов.

На этапе планирования (plan) в цикле PDSA вы наблюдаете за проблемой или открывшейся возможностью улучшить процесс и собираете данные для количественной оценки. Затем анализируете их, чтобы найти первопричины низкой эффективности, и формулируете гипотезу о взаимосвязи между первопричинами и наблюдаемым эффектом. После этого разрабатываете план эксперимента или изменений, которые помогут устранить проблему или воспользоваться возможностью. Именно так вы заполняли отчет A3: обозначили проблему в разделе «Тема», провели исследование в пункте «Предпосылки» и собрали данные в разделе «Текущее состояние». Этап планирования также охватывает раздел «Анализ первопричин», в котором вы выдвинули гипотезу о причинно-следственных связях, и «Контрмеры», где разработали эксперимент для ее проверки. На этапе выполнения (do) внедряются изменения или проводится небольшой эксперимент для проверки гипотезы. Это соответствует реализации разработанных контрмер. На этапе изучения (study) вы собираете и оцениваете результаты, чтобы проверить, подтверждают ли они гипотезу, а также фиксируете приобретенные знания о работе системы. Таким образом, этап соответствует разделу «Подтверждение результата» и способствует непрерывному обучению.

https://liveinternet.club

Наконец, на этапе действия (act) вы масштабируете изменения и делитесь приобретенными знаниями. Если изменение не принесло ожидаемых результатов, можно отказаться от него или перезапустить цикл. Этот этап соответствует разделу «Последующие действия» в отчете по методу A3. Цикл PDSA показан на рис. 2.14.

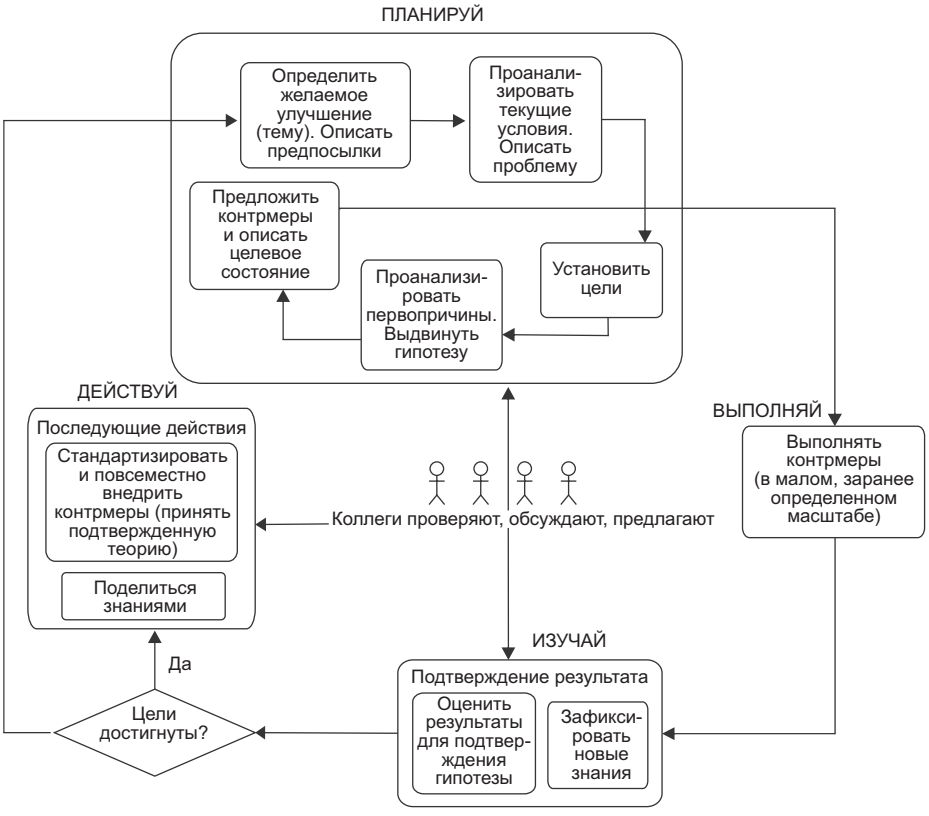


Рис. 2.14. Цикл PDSA в отчете A3

ПРИМЕЧАНИЕ Главное отличие между циклами PDSA и PDCA заключается в фокусе внимания при анализе результатов: в PDSA вы переосмысливаете саму гипотезу о работе системы, а в PDCA — только проверяете и корректируете план действий. На этапе проверки (check) в цикле PDCA вы определяете, успешно ли реализуется план. Если ожидаемый результат не достигнут, нужно вернуться на этап планирования и внести изменения. Подход в цикле PDSA несколько иной. У. Э. Деминг считал, что важно проработать не только план внедрения улучшений, но и саму теорию о том, как работает система. Он предлагал сначала спрогнозировать результаты изменения системы (сформулировать гипотезу), затем изучить фактические результаты нововведений и, наконец, сравнить их с первоначальными прогнозами, чтобы пересмотреть гипотезу

https://liveinternet.club

при необходимости. У. Э. Деминг подчеркивал, что новые знания о работе системы приобретаются только тогда, когда выдвинута гипотеза, которую можно подвергнуть сомнению и усовершенствовать, учитывая данные, полученные на этапе выполнения. Другими словами, чтобы по-настоящему понять систему, при нововведениях нужно опираться на *опровергаемую* гипотезу — такую, которую можно подтвердить или опровергнуть. Подробнее о цикле У. Э. Деминга можно прочитать в его книге «The New Economics for Industry, Government, Education»[1] (3-е издание, MIT Press, 2018).

Как видите, цикл PDSA очень напоминает научный метод исследования, так как включает в себя определение проблемы, ее изучение, сбор данных, разработку гипотезы, ее проверку, анализ результатов на соответствие гипотезе и формулировку заключения.

Таким образом, метод А3 не только развивает логическое мышление — он учит наблюдать за процессом и опираться на факты, а не на личные мнения. Благодаря отслеживанию количественных метрик процесса, отраженных в карте потока ценности (таких, как время реализации, время выполнения задачи и процент завершенных и корректных задач), можно избежать иллюзий и провести анализ объективно.

2.12.2. Достижение согласованности и налаживание совместной работы

Совместное ревью отчета в формате А3 с коллегами, которые вовлечены в создание потока ценности, помогает прийти к консенсусу и выработать согласованную стратегию для улучшений. У коллег появится возможность наблюдать за потоком вместе с вами, ознакомиться с представленным процессом и данными, предложить собственные идеи и участвовать в ревью отчета. Излагая информацию объективно и логично, вы сможете заручиться поддержкой внедрения изменений от всех стейкхолдеров.

2.12.3. Применение системного мышления

Разрабатывая отчет А3, вы учитесь смотреть на проблему целостно, с позиции системного мышления, и разрабатывать контрмеры, которые помогут ее устранить. Системное мышление — способ анализа, при котором система рассматривается как единое целое с учетом всех взаимосвязей между ее компонентами, а не как набор разрозненных элементов. Вместо того чтобы разбирать проблему по частям, вы изучаете рабочий процесс целиком, что может привести к совершенно иным выводам. В отчете А3 вы учитываете не только саму проблему, но и ее предпосылки, цели и приоритеты организации, мнение коллег и предложенные контрмеры, а анализ первопричин помогает лучше понять поведение системы.

[1] У. Э. Деминг. «Новая экономика».

https://liveinternet.club

Другие методы решения проблем

В этой книге я рекомендую внедрять APIOps, опираясь на метод A3, поскольку он широко используется специалистами в области бережливого производства. Однако важно понимать, что A3 — не единственный метод совершенствования процессов. Существуют и другие, например цикл DMAIC (define, measure, analyze, improve, control: определение, измерение, анализ, совершенствование, контроль) из концепции «Шесть сигм» и методика 8D (8 disciplines, 8 дисциплин), разработанная компанией Ford. Эти подходы основаны на логике цикла PDSA, но их обсуждение выходит за рамки моей книги. Подробнее о цикле DMAIC вы можете узнать на странице https://mng.bz/eoBz, а о методике 8D — на странице https://quality-one.com/8d.

2.13. Управление внедрением APIOps

Прежде чем завершить эту главу, я хотел бы дать вам несколько общих рекомендаций о том, как внедрять изменения. Переход организации к APIOps сопровождается большими переменами, а крупным компаниям любые перемены всегда даются непросто. Джон П. Коттер (John P. Kotter) в книге «Leading Change»[1] (Harvard Business Review Press, 2012) описал процесс управления изменениями, состоящий из восьми этапов. Рассмотрим, как можно применить их для внедрения APIOps.

1. *Убедите остальных в безотлагательной необходимости изменений.* Изучите рынок и предложения конкурентов, чтобы понять, какие угрозы, кризисы и возможности стоят перед вашей организацией. Например, как быстро другие компании поставляют код и API на продакшен? Опережают ли ваши API конкурентов с точки зрения согласованности, качества дизайна и полноты документации? Соберите информацию и покажите руководству и коллегам, что изменения необходимы уже сейчас. Продумайте наглядные сценарии того, к чему может привести бездействие. Убедите руководителей в том, что сохранять текущее положение дел опасно. Помните: нужно приложить усилия, чтобы вывести людей из зоны комфорта.
2. *Соберите сильную команду реформаторов.* Создайте группу из влиятельных сторонников изменений (так называемую «команду реформаторов»), объединенных общими целями и обладающих достаточными полномочиями, чтобы продвигать внедрение APIOps. Эта команда должна работать как единое целое, независимо от привычной иерархии внутри компании. По возможности включите в группу старших менеджеров продукта API, архитекторов API и спонсора проекта из числа высшего руководства, который готов поддержать инициативу и обеспечить необходимыми ресурсами, — им

[1] Дж. П. Коттер. «Впереди перемен».

https://liveinternet.club

может выступать руководитель подразделения или директор, отвечающий за программу API.

3. *Опишите свое видение трансформации.* Сформулируйте представление о будущем, которого вы хотите достичь путем внедрения APIOps, и кратко изложите свое видение, чтобы окружающие могли легко представить, к чему вы стремитесь. Определите, какие ценности будут направлять процесс изменений, и разработайте стратегии для воплощения идеи. Избегайте слишком абстрактных или сложных формулировок. Важно, чтобы команда реформаторов могла объяснить ваше видение не более чем за 5 минут.
4. *Донесите свое видение до участников.* Используйте каждую возможность, чтобы рассказать о новом видении и выбранной стратегии. Не допускайте, чтобы люди забыли о переменах, — говорите о них чаще. Кроме того, команда реформаторов должна своим поведением демонстрировать приверженность новым идеям. Регулярно обсуждайте свое видение APIOps на брифингах и встречах профессионального сообщества компании.
5. *Устраняйте препятствия и открывайте возможности для сотрудников.* Устраните все препятствия — будь то устаревшие системы или структуры, которые мешают воплотить вашу идею. Проанализируйте используемые инструменты и откажитесь от тех, что затрудняют переход к APIOps. Вместо них предоставьте коллегам решения, которые соответствуют принципам APIOps, а также помогают внедрять инновации и налаживать взаимодействие между подразделениями.
6. *Создайте условия для быстрых побед.* Чтобы вдохновить команду на перемены, запланируйте достижимые цели и уже в первые недели продемонстрируйте рост производительности. В зависимости от проблем, с которыми столкнулась ваша организация, для начала отлично подойдут внедрение линтинга API (глава 3), выявление критических изменений (глава 4) и создание простого пайплайна для обработки определений API (глава 8). Не забывайте отмечать вклад коллег, которые помогают внедрять улучшения.
7. *Закрепляйте успехи.* Добившись первых успехов, не торопитесь объявлять полную победу. Вместо этого закрепите результаты, чтобы использовать их как фундамент для более масштабных изменений в организации, включая обновление систем и структур. К примеру, подумайте: как интегрировать APIOps в процессы разработки веб-API или даже создание API для обмена сообщениями. Кроме того, следует нанимать, повышать и обучать именно тех сотрудников, которые разделяют ваш подход и смогут воплощать его в будущем.
8. *Делайте изменения частью корпоративной культуры.* По возможности демонстрируйте, как внедренные принципы и методы APIOps способствуют общему успеху компании. Убедитесь, что потенциальные сотрудники

https://liveinternet.club

знакомы с вашими подходами к DevOps и APIOps. Создайте или обновите внутренние курсы разработки API, чтобы все коллеги, независимо от стажа работы в компании, могли освоить и применять принципы APIOps в повседневной работе.

Итоги

- При внедрении DevOps и активной разработке продуктов API в организации возникает проблема масштабирования: становится все труднее поддерживать возросший темп проектирования, документирования и развертывания API.
- Возникают следующие проблемы: увеличивается время реализации API, падает качество документации и нарушается единообразие интерфейсов. Оптимальный момент для внедрения APIOps наступает, когда в организации осознают перечисленные проблемы и стремятся их решить.
- Метод A3 для решения проблем помогает не только фиксировать идеи и формировать отчеты, но и внедрять APIOps в соответствии с научным подходом. С помощью отчета в формате A3 можно спланировать улучшение процесса, согласовать действия с коллегами, проверить, приводят ли изменения к ожидаемым результатам, и продемонстрировать всем достигнутый прогресс.
- Составление карт процессов при помощи техник, принятых для систематизации потока ценности, — отличный способ визуализировать процесс, требующий автоматизации в рамках APIOps. Карты позволяют выявить потери, которые можно устранить, и значимые действия, которые можно оптимизировать.
- На первых этапах внедрения APIOps следует показать необходимость безотлагательных изменений, создать команду реформаторов, способных инициировать эти изменения, и описать свое видение будущего процесса.

https://liveinternet.club

3 Линтинг API: автоматизация единообразия API

В этой главе

- ✓ Как с помощью линтинга обеспечить соответствие API руководству по стилю
- ✓ Написание собственных правил линтинга API
- ✓ Создание правил линтинга с помощью ИИ

В главе 1 я познакомил вас с подходом APIOps и его преимуществами, в главе 2 поделился основными инструментами для его внедрения. Пора перейти к настоящей практике и сделать первый шаг в работе с APIOps. В этой главе вы узнаете, как запустить правила линтинга API, чтобы проверить, соответствует ли разработанный интерфейс руководству по стилю. Материал особенно пригодится разработчикам, которые проектируют и внедряют API, и может оказаться полезен владельцам и менеджерам продуктов API, желающим получить общее представление о том, как работает линтинг. Я научу вас запускать линтинг API в редакторе и через командную строку, а также покажу, как создать простое правило. Кроме того, вы узнаете, как упростить создание правил с помощью ИИ.

Прежде чем приступить к работе, установите необходимые инструменты по инструкциям из приложения Б. Вам понадобятся Spectral CLI, Visual Studio Code (VS Code), плагин OpenAPI (Swagger) для редактора VS Code, yq, jpp и GitHub

https://liveinternet.club

Copilot[1]. Чтобы выполнять упражнения, клонируйте репозиторий книги (https://github.com/apiopsbook/apiops) и перейдите в папку `chapter3`. Если вы просто хотите ознакомиться с результатами, откройте папку `chapter3-completed` — в ней вы найдете выполненные задания.

3.1. Линтинг API в APIOps

На этапе проектирования продукта API необходимо убедиться, что дизайн соответствует руководству по стилю. Оно служит справочным документом, в котором зафиксированы различные соглашения, например правила именования API, паттерны разбивки на страницы, форматы ошибок, шаблоны путей и другие решения, следуя которым специалисты организации могут привести API к стандартизированному виду. Благодаря стандартизации удается поддерживать единообразие API, что, в свою очередь, повышает безопасность и удобство использования интерфейса, а также способствует его распространению. Руководство по стилю (также «стандарт» или «руководство по проектированию») является важным инструментом для управления API. Обычно в нем изложены правила, которым должен соответствовать дизайн. К примеру, может быть указано, что пути в URI записываются в «верблюжьем регистре», для аутентификации используется OAUth2, а сообщение об ошибке должно содержать ее описание и код. Как правило, организации, в которых уже налажены процессы управления API, используют собственные руководства по стилю. Но если вы только начинаете разрабатывать API, можно обратиться к общедоступным версиям документов и адаптировать их под собственные нужды.

Примеры руководств по стилю API

Вот несколько распространенных общедоступных руководств по стилю для REST API:

- Zalando RESTful API and Event Guidelines: https://mng.bz/aE9z;
- Microsoft REST API Guidelines: https://mng.bz/d6KN;
- Haufe-Lexware API Style Guide: https://mng.bz/rV4B;
- REST Guidelines от Belgian Interoperability Framework: https://mng.bz/Vx4P;
- подборка общедоступных руководств по стилю API, сгруппированных по темам, от Арно Лоре (Arnaud Lauret): http://apistylebook.com.

Линтинг API — это автоматизированная проверка файлов определения API на соответствие руководству по стилю. Он похож на статический анализ кода, только применяется к файлам определения API. Инструменты линтинга быстро сообщают разработчику, насколько API соответствует стандарту, описанному в справочных материалах. Это избавляет от необходимости читать и запоминать

[1] Хотя автор приводит примеры на GitHub Copilot, описанные практики применимы к Cursor и другим современным ИИ-средам разработки. Использование ИИ-инструментов значительно ускоряет создание правил линтинга, но требует обязательной ручной проверки из-за возможных галлюцинаций и ошибок ИИ.

https://liveinternet.club

десятки страниц документации и тем самым повышает эффективность работы. На этапе проектирования разработчики запускают линтинг определений API локально через IDE или командную строку, а также при сохранении изменений в процессе непрерывной интеграции. Этап продемонстрирован на рис. 3.1.

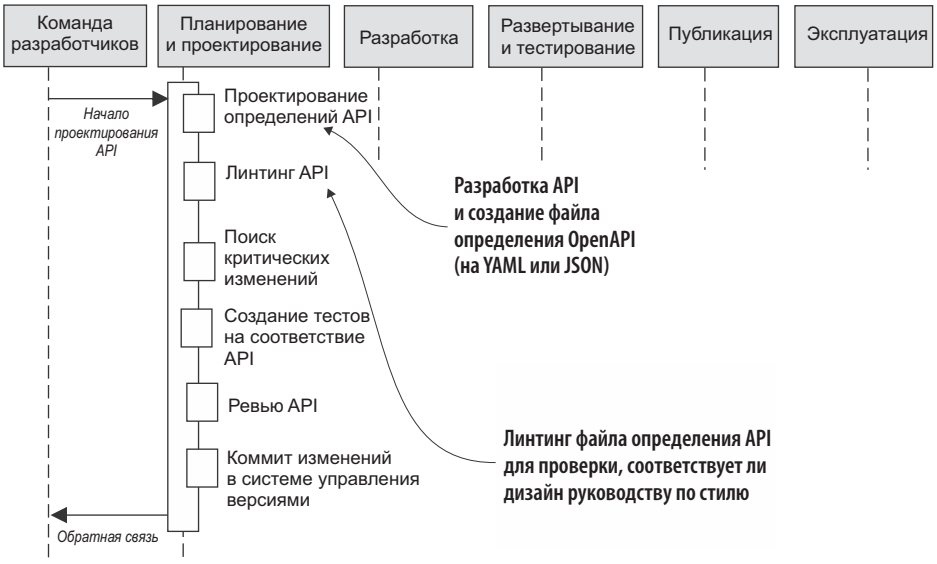


Рис. 3.1. Запустите линтинг API на этапе проектирования, чтобы узнать, насколько API соответствует руководству по стилю

3.2. Проблема соблюдения требований в API

Представьте, что команде разработчиков поручили спроектировать API в соответствии со стандартом JSON:API (https://jsonapi.org). Он предназначен для разработки REST API и определяет, каким образом клиент должен запрашивать ресурсы для их извлечения или изменения, а сервер — отвечать на эти запросы. JSON:API представляет собой подробную спецификацию с четкими правилами о том, как форматировать JSON-ответы REST API, как оформлять сообщения об ошибках, какие типы данных использовать в запросах и ответах, и другими соглашениями.

REST, JSON:API и OAS

REST (Representational State Transfer) — архитектурный стиль веб-сервисов, предложенный Роем Филдингом (Roy Fielding). Филдинг определил, что API, соответствующие этому стилю, должны работать без сохранения состояния, обеспечивать независимость клиента и сервера, использовать единый интерфейс, поддерживать кэширование и при необходимости предоставлять исполняемый код. REST API позволяет обращаться к ресурсу по его унифицированному указателю (Uniform Resource Locator, URL) и взаимодействовать с этим ресурсом на сервере с помощью HTTP-глагола, например `GET`, `POST` или `PATCH`.

https://liveinternet.club

В этой книге при описании REST API используются следующие термины.

- *Ресурс* — объект, обладающий типом, связанными данными (набором атрибутов) и отношениями с другими ресурсами. Например, в запросе `GET /users/1234` ресурсом является пользователь с уникальным идентификатором 1234.
- *Коллекция* — группа однотипных ресурсов. При этом сами коллекции также считаются ресурсами. Пример запроса коллекции: `GET /users`.
- *Вложенный ресурс* — ресурс, который находится в иерархии ниже другого ресурса и связан с ним отношениями «родитель — потомок». Например, запрос `GET /users/1234/article/9876` вернет вложенный ресурс article 9876, принадлежащий родительскому ресурсу user 1234.

OAS (OpenAPI Specification) — формат описания любого веб-API (в том числе REST API), работающего по протоколу HTTP. Определяет, как описывать HTTP API с помощью определения на YAML или JSON. В свою очередь, JSON:API — строгая спецификация, которая задает правила проектирования REST API. Таким образом, API, разработанный по стандарту JSON:API, можно описать в формате OAS.

Команда разработчиков изучила спецификацию JSON:API и создала API, опираясь на ее предписания. Специалисты считают, что API соответствует требованиям, но все-таки не до конца в этом уверены. Вас попросили проверить, действительно ли в дизайне соблюдается стандарт JSON:API. Как убедиться в этом с помощью линтера?

ПРИМЕЧАНИЕ Особое внимание в APIOps уделяется единообразию: API обязательно должен соответствовать утвержденному руководству по стилю. Хотя в книге я упоминаю некоторые аспекты проектирования API, подробное обсуждение этой темы выходит за ее рамки. Если вы хотите глубже ознакомиться с методами проектирования API, рекомендую две отличные книги: «Principles of Web API Design» (Addison-Wesley Professional, 2021) Джеймса Хиггинботама (James Higginbotham) и «The Design of Web APIs»[1] (Manning, 2019) Арно Лоре.

Конечно, проверить API можно и без линтера — просто прочитайте всю спецификацию JSON:API и самостоятельно проанализируйте каждый запрос и ответ в файле определения API. Однако в спецификации JSON:API так много правил, что возникает риск упустить некоторые из них. Поэтому научимся автоматизировать проверку с помощью линтера API.

3.3. Знакомство с линтерами API

Для работы инструментов линтинга API требуется файл определения, который нужно проверить, и набор правил, которым он должен соответствовать. На выходе линтер выдает список выявленных проблем с указанием строки в файле определения и нарушенного правила. Линтеры API можно интегрировать в редакторы API или использовать отдельно через командную строку.

[1] А. Лоре. «Проектирование веб-API».

https://liveinternet.club

Наиболее распространенным линтером OpenAPI с открытым исходным кодом считается Spectral от Stoplight (https://github.com/stoplightio/spectral), который поддерживает файлы JSON и YAML. Он встроен в инструменты для проектирования и совместной разработки API, включая Stoplight Studio, Postman, Insomnia и др. Кроме того, Spectral доступен в виде плагинов для многих IDE и текстовых редакторов, в том числе IntelliJ IDEA и VS Code. Среди альтернатив стоит отметить Redocly CLI (https://github.com/Redocly/redocly-cli) — линтер API, отличающийся простой работой с файлами OpenAPI. Также существует Vacuum (https://github.com/daveshanley/vacuum) — быстрый линтер OpenAPI, совместимый с набором правил Spectral. Наконец, интересным решением стал инструмент Кевина Свайбера (Kevin Swiber) под названием Spego (https://github.com/kevinswiber/spego), в котором для проверки определений OpenAPI используется OPA (Open Policy Agent). В Spego правила пишутся на декларативном языке Rego, применяемом для определения политик в OPA.

В этой главе я буду выполнять линтинг API через командную строку с помощью Spectral CLI. Вы также узнаете, как использовать плагин Spectral для линтинга в VS Code — одном из самых распространенных текстовых редакторов среди тех, кто работает с файлами определения API.

Нужен ли линтинг небольшим компаниям, в которых нет руководства по стилю API?

Как правило, небольшие организации, которые только начинают разрабатывать API, еще не сформировали собственное руководство по стилю. Стоит ли вообще проводить линтинг API, если документа с правилами пока нет?

Подготовка руководства по стилю API занимает немало времени. Хотя в интернете много отличных примеров, их все равно приходится адаптировать под собственные нужды. Тем не менее даже если набор правил еще не сформирован, линтер уже может выявить некоторые ошибки в файле определения OpenAPI. Во-первых, он проверяет корректность файла определения OpenAPI. Это важно, если вы генерируете документацию на его основе, делитесь им с внешними потребителями или передаете в качестве исходных данных в какой-либо инструмент. Во-вторых, такие линтеры, как Spectral и Redocly CLI, содержат встроенные наборы базовых правил, применимых к большинству определений API. Например, инструменты могут проверить, соответствуют ли примеры объектов, указанные в файле определения, объявленной схеме. Поэтому я советую организациям, где всего одна-две команды разработчиков, внедрять линтинг API и применять встроенные правила как можно раньше. Со временем вы сформируете собственное руководство по стилю и сможете дополнить им стандартные правила линтеров.

3.4. Проверка на соответствие требованиям с помощью Spectral

Вернемся к сценарию из раздела 3.2 и используем Spectral CLI, чтобы проверить, соответствует ли файл определения API, полученный от разработчиков, набору правил JSON:API. Обычно набор содержит правила, написанные на JSON, YAML

https://liveinternet.club

или JavaScript на основе руководства по стилю. В папке `chapter3` из клонированного репозитория вы найдете файл `json-api.ruleset.yaml` с правилами из спецификации JSON:API. Там же находится `product-catalog.oas.yaml` — файл определения API, подготовленный вымышленной командой. Выполните команду линтинга Spectral с параметром `verbose`:

```
$ spectral lint product-catalog.oas.yaml
➥ --ruleset json-api.ruleset.yaml --verbose
```

Вы получите следующий результат:

```
Found 23 rules (23 enabled)
Linting /Users/ikenna/chapter2/product-catalog.oas.yaml
No results with a severity of 'error' found!
```

Spectral проверил определение API на соответствие набору правил и не нашел проблем. Отлично! Значит, в файле определения API, полученном от команды, соблюдаются все правила, а вам даже не пришлось читать и анализировать спецификацию JSON:API, чтобы это выяснить.

А если бы в API действительно была ошибка? Проверим другой файл — `product-catalog-v2.oas.yaml`, в котором нарушено правило. Выполните команду `lint`, чтобы узнать, что произойдет.

```
$ spectral lint product-catalog-v2.oas.yaml
➥ --ruleset json-api.ruleset.yaml
➥ --verbose
```

Вы увидите подобный вывод:

```
Found 23 rules (23 enabled)
Linting /Users/ikenna/chapter3/product-catalog-v2.oas.yaml
/Users/ikenna/chapter3/product-catalog-v2.oas.yaml
41:30 error response-content-type All JSON:API response bodies MUST
➥ be returned with the header Content-Type: application/vnd.api+json
➥ paths./v1/catalog/categories.get.responses[200].content.
➥ application/json
```

Spectral обнаружил ошибку в строке 41. Согласно спецификации JSON:API, все ответы должны возвращаться с заголовком Content-Type `application/vnd.api+json`, но в определении ответа для операции `GET /v1/catalog/categories` этот заголовок не указан.

Вот как выглядит фрагмент с ошибкой (строки 40–43):

```
content:
  application/json:
    schema:
      $ref: '#/components/schemas/CategoriesResponse'
```

В строке 40 указано application/json вместо application/vnd.api+json

Теперь еще раз взгляните на отчет Spectral. Вы увидите, что он содержит необходимую информацию: сначала указываются строка и позиция (`41:30`),

https://liveinternet.club

затем — уровень серьезности проблемы (`error`), имя нарушенного правила из набора (`response-content-type`), пояснение ошибки (`All JSON:API response bodies MUST be returned…`) и, наконец, ее расположение (`paths./v1/catalog/categories.get.responses[200].content.application/json`).

3.5. Использование Spectral в VS Code

Теперь нам понадобится редактор VS Code и расширение Spectral — если вы следовали инструкциям из приложения Б, они должны быть установлены и готовы к работе. В настройках расширения можно задать собственный набор правил для проверки файла определения API. Для этого откройте настройки VS Code и найдите расширение Spectral. В поле раздела `Spectral:Ruleset` укажите `json-api.ruleset.yaml`. Откройте в редакторе файл определения API `product-catalog-v2.oas.yaml`. Вы заметите, что VS Code подсвечивает проблемные строки, начиная с номера 41. Чтобы увидеть описание проблемы, наведите курсор на одну из выделенных строк. Сообщение об ошибке можно также найти внизу, на вкладке Problem (рис. 3.2).

Благодаря интеграции линтера Spectral в редактор API вы уже на этапе проектирования можете узнать, соответствует ли файл определения API правилам. В этой книге я использую VS Code в паре с плагином OpenAPI (Swagger) компании 42Crunch, но есть и другие распространенные редакторы OpenAPI. Многие из них представляют собой инструменты для проектирования, совместной разработки и ревью API: Stoplight, Postman, Apicurio, SwaggerHub и Insomnia. Некоторые предлагают удобные визуальные интерфейсы для редактирования определений OpenAPI, комментирования, линтинга API и создания имитаций.

3.6. Создание правила Spectral для поля info.description

В предыдущих разделах вы использовали в Spectral готовые правила из набора. Но как написать собственное правило на основе руководства по стилю, утвержденного в организации? Допустим, вам поручили убедиться, что в объекте `info` каждого определения OpenAPI есть поле `description`. В результате линтер должен сообщить, что поле `$.info.description` отсутствует в следующем фрагменте определения OpenAPI:

```
openapi: 3.0.3
info:
  title: API каталога продуктов Acme
  version: 0.0.1
paths: {}
```

Как этого добиться? Прежде всего нужно понять, как устроено правило для проверки в Spectral. Набор правил можно задать в файле JSON, YAML или JavaScript; в этой главе вы будете использовать YAML. Файл для Spectral должен содержать поле верхнего уровня `rules`, в котором хранится собственно массив правил. Каждое правило имеет уникальное имя и несколько свойств. В нашем случае достаточно четырех: `message`, `given`, `then` и `severity`.

https://liveinternet.club



Рис. 3.2. Использование собственного набора правил линтинга и просмотр ошибок в VS Code

Свойство `message` определяет текст сообщения, которое выводится при нарушении правила линтинга. Вам необходимо понятно объяснить пользователю суть проблемы. В нашем случае можно сообщить, что файлы определения API должны содержать поле `description`.

https://liveinternet.club

Свойство `given` представляет собой одно или несколько выражений на языке JSONPath, указывающих, к каким частям документа применяется правило. Выражение может вернуть любое количество элементов, и если оно возвращает ноль — правило не проверяется. В нашем случае свойство `given` может представлять собой выражение `$.info`, которое возвращает весь объект `info`. Далее с помощью свойства `then` мы задаем функцию, проверяющую, содержит ли выбранный объект поле `description`.

> **ПРИМЕЧАНИЕ** JSONPath — язык запросов, предназначенный для работы с файлами JSON и во многом похожий на XPath, который используется для XML. С помощью выражений на JSONPath можно извлекать данные из документов JSON. Все выражения начинаются с символа $, обозначающего корневой элемент документа. Далее с помощью точек (*точечная нотация*) или квадратных скобок (*скобочная нотация*) указываются дочерние элементы. Например, если взять показанный фрагмент OpenAPI и представить его в формате JSON, то выражение на JSONPath `$.info.version` вернет значение `0.0.1`. Подробную информацию о выражениях на JSONPath вы найдете на странице https://github.com/json-path/JsonPath.

В свойстве `then` происходит проверка правила: оно применяется к частям документа, найденным с помощью выражения на JSONPath. У этого свойства есть два ключевых слова: `function` и `field`.

Ключевое слово `field` необязательно и используется, чтобы точно обозначить, к какому свойству объекта следует применить функцию. В нашем случае ключевое слово `field` будет иметь значение `description`, поскольку нужно убедиться, что именно это поле содержится в объекте `info`.

Ключевое слово `function` позволяет задать имя функции, которую Spectral выполнит для выбранного объекта. Она сообщит о нарушении правила, только если обнаружит проблему с объектом. Некоторые функции уже определены в Spectral, но при необходимости вы можете создать и собственные на JavaScript или TypeScript. В нашем примере подойдет стандартная функция `truthy`. Она проверяет, что объект не принимает значение `false`, `0`, `null` и `undefined` и не является пустой строкой. Так что, если поле `info.description` вернет `null`, функция сработает и сообщит о нарушении правила.

Наконец, свойство `severity` характеризует уровень серьезности правила, что позволяет группировать и фильтровать нарушения по их важности. При запуске Spectral через командную строку можно также использовать флаг `--fail-severity`, чтобы задать уровень серьезности нарушения, начиная с которого команда будет завершаться с ошибкой. Таких уровней в линтере предусмотрено пять, в порядке от высшего к низшему: `error`, `warn`, `info`, `hint` и `off`. На рис. 3.3 показана структура свойств внутри набора правил.

Четыре свойства правила (`message`, `given`, `then` и `severity`) показаны на рис. 3.4 в виде упрощенной синтаксической диаграммы.

https://liveinternet.club



Рис. 3.3. Структура простого правила из набора Spectral. В свойстве `given` указывается часть файла определения API, к которой применяется правило, а само правило описано в свойстве `then`

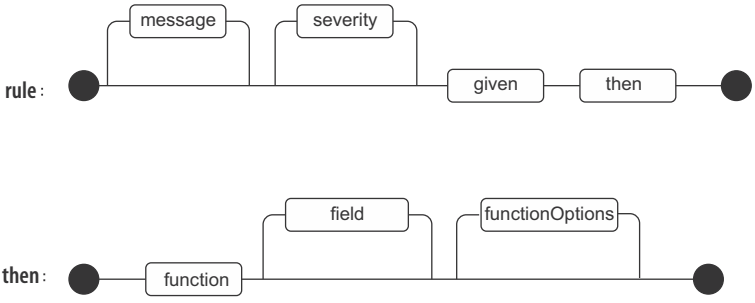


Рис. 3.4. Упрощенная синтаксическая диаграмма для правила Spectral. Хотя обычно правила содержат больше элементов и петель, чем показано на этой диаграмме, она наглядно представляет рассматриваемый пример

Итак, вы узнали, как создавать простые правила Spectral, и теперь можете проверить, определено ли поле `$.info.description`. Найдите в проекте файл `simple.`

https://liveinternet.club

ruleset.yaml и откройте его в VS Code или другом текстовом редакторе. Внесите в файл код из листинга 3.1 и сохраните его[1].

Листинг 3.1. Правило Spectral для проверки наличия поля info.description

```
rules:
  info-description-is-required:
    message: " Файлы определения API должны
➥ содержать поле info.description"
    severity: error
    given: $.info
    then:
      field: description
      function: truthy
```

Имя правила

Сообщение для вывода при нарушении правила

Уровень серьезности правила

Выражение на JSONPath, определяющее элемент документа, к которому должно применяться правило

Поле элемента, которое оценивается с помощью функции Spectral

Функция truthy в Spectral, которая сработает, если $.info.description примет значение null

Проверьте новое правило на файле определения API, в котором отсутствует элемент info.description. Выполните следующую команду в командной строке:

```
$ spectral lint minimal.oas.yaml --ruleset simple.ruleset.yaml
```

В результате вы получите:

```
2:6 error info-description-is-required API definition files must have
➥ info.description defined info
✖ 1 problem (1 error, 0 warnings, 0 infos, 0 hints)
```

Вы только что впервые создали и применили правило Spectral. Вам понадобилась функция truthy, которая оценивает значение, возвращаемое выражением на JSONPath. Помимо нее Spectral предлагает и другие встроенные функции. Так, length подсчитывает количество символов в строке, элементов в массиве и свойств в объекте, а также проверяет, укладываются ли полученные значения в заданные пределы; casing проверяет, соответствует ли текст выбранному регистру (например, «верблюжьему регистру» или «кебаб-регистру»); pattern позволяет сопоставлять регулярные выражения. Полный список встроенных функций Spectral доступен на странице https://mng.bz/gvml.

В приведенном примере функция truthy работала без дополнительных параметров (functionOptions), которые позволяют передавать функции Spectral необходимые входные значения. В следующем разделе вы узнаете, как использовать functionOptions.

[1] При создании правила можно задать текст сообщения (message) на любом языке, включая русский. Однако на практике так делают очень редко (почти никогда). Для удобства читателя описания правил и тексты сообщений переведены на русский язык, но следует иметь в виду сказанное выше. — *Примеч. науч. ред.*

https://liveinternet.club

3.7. Создание правила Spectral для проверки версионирования путей

Представьте, что вам поручили реализовать правило из руководства по стилю API, требующее, чтобы каждый путь URI начинался с номера версии. Это означает, что в файле определения API нужно выбрать все URI и проверить, начинаются ли они с шаблона `/v{digit}/`, где `{digit}` представляет собой номер. Допустим, вы хотите опробовать новое правило на файле определения `product-catalog-v3.oas.yaml`.

При создании правила Spectral нужно указать в свойстве `given` выражение на JSONPath, которое выбирает все URI в определении API. Однако базовый синтаксис JSONPath не позволяет сделать это напрямую. К счастью, Spectral поддерживает расширение JSONPath Plus, содержащее более полный набор выражений. Чтобы получить все пути в документе, вам понадобится выражение `$.paths[*]~` на JSONPath Plus. Подробную информацию о выражениях JSONPath Plus вы найдете на странице https://github.com/JSONPath-Plus/JSONPath.

Проверка выражений JSONPath Plus

Предположим, вы хотите убедиться, что указанное выше выражение JSONPath Plus найдет четыре URI в файле определения API `product-catalog-v3.oas.yaml`. Для этого нужно сначала преобразовать файл `product-catalog-v3.oas.yaml` в формат JSON, а затем протестировать выражение JSONPath Plus на полученном документе. Преобразовать файл YAML в JSON можно в онлайн-сервисах или с помощью инструмента командной строки yq, установка которого описана в приложении Б. Если yq готов к работе, выполните в нем следующую команду для преобразования `product-catalog-v3.oas.yaml` в `product-catalogv3.oas.json`:

```
$ yq eval -o=json product-catalog-v3.oas.yaml >
➥ product-catalog-v3.oas.json
```

Протестировать выражение JSONPath Plus на файле `product-catalog-v3.oas.json` можно двумя способами. Во-первых, воспользовавшись онлайн-инструментом тестирования, например JSON Query (www.jsonquerytool.com). Для этого вставьте содержимое файла `product-catalogv3.oas.json` в раздел Input на странице сайта, выберите JSONPath Plus в поле Transform и введите `$.paths[*]~` в разделе Query.

Во втором способе понадобится инструмент командной строки jpp, предназначенный для проверки выражений JSONPath Plus. Инструкции по его установке вы также найдете в приложении Б. Для проверки просто передайте выражение и файл JSON в качестве входных данных, и программа выведет список путей, соответствующих выражению:

```
$ jpp "$.paths[*]~" product-catalog-v3.oas.json
```

Далее необходимо определить свойство `then`, задав в нем функцию `pattern`. Это встроенная функция Spectral, которая проверяет совпадение значений

https://liveinternet.club

с регулярными выражениями. Чтобы она работала корректно, передайте ей регулярное выражение с помощью ключевого слова в `functionOptions`:

```
then:
  function: pattern
  functionOptions:
    match: '^/v\d+/.*$'
```

Для функции `pattern` в `functionOptions` предусмотрено два ключевых слова: `match` и `notMatch`. Слово `match` задает регулярное выражение, которому должен соответствовать проверяемый элемент. В свою очередь, `notMatch`, наоборот, подразумевает, что элемент не должен совпадать с заданным выражением.

При определении правила также можно указать свойство `description` — краткое описание, которое помогает понять, в чем заключается это конкретное правило.

Обратите внимание, что свойство `message` может включать плейсхолдеры, вместо которых при выполнении правила подставляются реальные значения. К примеру, плейсхолдер `{{description}}` будет заменен описанием правила, `{{property}}` — именем элемента в определении API, который вызвал нарушение, а `{{path}}` — полным путем к этому элементу.

Теперь, чтобы выполнить поставленную задачу, добавьте правило `uri-path-must-include-version` в файл `simple.ruleset.yaml`, как показано в следующем листинге.

Листинг 3.2. Добавление правила для проверки номера версии пути

```
rules:
  info-description-is-required:
    message: " Файлы определения API должны содержать поле info.description"
    severity: error
    given: $.info
    then:
      field: description
      function: truthy
  uri-path-must-include-version:
    description: Путь URI должен содержать номер версии
    message: '{{description}}; {{property}}
➥ должно начинаться с /v{digit}/'
    severity: error
    given: '$.paths[*]~'
    then:
      function: pattern
      functionOptions:
      match: '^/v\d+/.*$'
```

Имя нового правила

Короткое описание правила

Свойство message может содержать плейсхолдеры, принимающие значения во время выполнения

Выражение JSONPath Plus для проверки массива путей URI в определении API

Регулярное выражение, требуемое для работы функции pattern в Spectral, указывается с помощью ключевого слова в functionOptions

Теперь выполните правило с помощью следующей команды в командной строке:

```
$ spectral lint product-catalog-v3.oas.yaml
➥ --ruleset simple.ruleset.yaml
```

https://liveinternet.club

Вы должны увидеть сообщение Spectral о проблеме: `Путь URI должен содержать номер версии; /catalog/categories должен начинаться с /v{digit}/`.

Итак, вы создали еще одно правило линтинга в файле `simple.ruleset.yaml`. Теперь в наборе уже два правила, и вы можете добавить новые или даже импортировать правила, определенные в других наборах. Далее рассмотрим, как импортировать или дополнять наборы правил в Spectral.

3.8. Подключение набора правил

Spectral позволяет импортировать наборы правил из других источников с помощью свойства `extends`. Так можно пополнить собственный список правил за счет встроенных наборов Spectral, локальных файлов, наборов на HTTP-серверах или правил, определенных в NPM (Node Package Manager, менеджер пакетов Node). Чтобы импортировать набор правил, используйте свойство `extends`. Посмотрите, как оно работает, на примере изначально пустого файла `.spectral.yaml`, добавив в него код из следующего листинга, импортирующий встроенный набор правил Spectral для OAS (`oas`). Набор `oas` содержит общие правила, применяемые к определениям OpenAPI 2.x и 3.x, и пригодится для проверки файла определения OpenAPI.

Листинг 3.3. Подключение набора правил Spectral oas

```
extends: ["spectral:oas"]
```

Свойство extends принимает массив наборов правил для импорта. Здесь вы указываете только один набор правил — Spectral oas

Протестируйте набор правил на файле определения API `minimal.oas.yaml`, содержащем минимально необходимую структуру для соответствия `oas`:

```
$ spectral lint minimal.oas.yaml
```

Обратите внимание: если не указывать имя файла с набором правил, то Spectral ищет и применяет файлы `.spectral.yaml`, `.spectral.yml` или `.spectral.json` из текущей папки. Линтер должен найти набор файлов `.spectral.yaml` и затем сформировать подобный отчет:

```
1:1 warning oas3-api-servers OpenAPI "servers" must be present
➥ and non-empty array.
 2:6 warning info-contact    Info object must have "contact"
➥ object. Info
 2:6 warning info-description Info "description" must be present and
➥ non-empty string. Info
```

В отчете Spectral представлены три предупреждения (warning) и ни одной ошибки. Это означает, что файл определения API корректен. Полный список правил Spectral из набора `oas`, использованных во время проверки документа, можно увидеть в документации Spectral на странице https://mng.bz/ppZG.

https://liveinternet.club

В ходе импорта набора правил можно изменить уровень их серьезности. Допустим, вы не хотите выполнять правило `info-description` из импортированного набора `oas`. В таком случае измените файл `.spectral.yaml`, как показано в листинге 3.4.

Листинг 3.4. Выключение импортированного правила info-description

```
extends: ["spectral:oas"]
rules:
  info-description: off
```

Уровень серьезности импортированного правила принимает значение off

Снова выполните команду Spectral `lint`, выбрав файл `minimal.oas.yaml`. На этот раз в отчете будут показаны только две проблемы, а предупреждение `info-description` не появится.

Общедоступные правила линтинга в Spectral

Возможно, вы уже задавались вопросом, существуют ли общедоступные наборы правил Spectral, которые можно использовать в своем проекте, — да, существуют. Вот три из них.

- *Postman Governance Rules*. Рабочее пространство в Postman, где правила линтинга Spectral оформлены в виде коллекций, что позволяет легко создать их копию и сразу начать тестировать (https://mng.bz/OZwn).
- *APIs You Won't Hate: API Style Guide*. Фил Стерджен (Phil Sturgeon) собрал наборы правил, в основе которых лежат рекомендации из его книги «Build APIs You Won't Hate» (https://github.com/apisyouwonthate/style-guide).
- *Azure API Style Guide*. Репозиторий Microsoft Azure в GitHub, содержащий наборы правил, которые используются для проверки определений OpenAPI в сервисах Azure (https://github.com/Azure/azure-api-style-guide).

Наконец, предположим, что вы решили подключить набор правил, размещенный на HTTP-сервере. Для этого достаточно указать URI набора в свойстве `extends`, как показано в листинге 3.5. Обратите внимание, что Spectral поддерживает форматы YAML, JSON, JavaScript и TypeScript.

Листинг 3.5. Подключение дополнительных наборов правил

```
extends:
  - spectral:oas
  - https://unpkg.com/@apisyouwonthate/style-guide@1.3.2
➥ /dist/ruleset.js
 rules:
  info-description: off
```

Добавляет URI на внешний набор правил для импорта

Снова выполните команду Spectral `lint`, выбрав файл `minimal.oas.yaml`, и вы увидите два новых сообщения о проблемах (`api-health` и `api-home`).

https://liveinternet.club

3.9. Переопределение правил линтинга API

Хорошо, если у вас есть полный набор правил линтинга, который можно сразу применить к новому проекту API. Они помогают разрабатывать качественный API уже на этапе проектирования, когда код еще не написан и внести изменения в случае ошибок будет легко. Но как быть, если проект API уже работает в продакшен-среде, но его код не соответствует набору правил линтинга? Представьте, что вам поручили применить правила к API, который отличается от последней версии руководства по стилю. Как обеспечить автоматическое соблюдение руководства?

Существуют три способа решения этой задачи. Первый — добавить набор правил в проект и обновить API и код проекта так, чтобы они этим правилам соответствовали. Это может значительно повлиять на структуру файла OpenAPI и исходный код проекта. Более того, для приведения API в соответствие с набором правил могут потребоваться критические (обратно несовместимые) изменения (подробнее о них — в главе 4). Подобный подход «большого взрыва» не только потребует немало усилий, но и может нарушить работу текущих пользователей API.

Второй способ — создать отдельные наборы правил для каждого проекта и обеспечить соответствия этим наборам. Вы формируете собственный набор для существующего проекта, содержащий только те правила, которые в нем выполняются. Такой подход гарантирует, что дальнейшие изменения API будут соответствовать хотя бы минимальным требованиям. Однако при этом теряется возможность автоматизировать проверку тех правил, которые вы не включили в набор, но которые были бы полезны для проверки новых элементов дизайна API. В итоге вы получаете несколько наборов правил.

Третий способ — использовать один и тот же набор правил во всех проектах, но сами правила настраивать и применять постепенно. В этом случае вы можете включить только те проверки, которые уже выполняются, а остальные — временно отключить или понизить их уровень серьезности для определенных объектов OpenAPI. Постепенное приведение к соответствию позволяет автоматически проверять, соблюдены ли текущие стандарты в новых версиях API. Затем, когда у вас появится время и ресурсы, вы сможете вернуться к отключенным правилам и подумать, получится ли изменить API так, чтобы он им соответствовал. Вероятно, некоторые правила для эндпоинтов так и останутся отключенными, но, по крайней мере, вы обеспечите соответствие там, где это возможно.

Итак, метод постепенного приведения к соответствию зависит от возможности отключать определенные правила линтинга или настраивать их серьезность для выбранных объектов в документе OpenAPI. Значит, необходимо сообщить линтеру, какие именно правила и для каких документов и объектов следует отключить. Для этого Spectral использует механизм под названием *переопределения* (overrides). Он позволяет адаптировать один и тот же набор правил под разные проекты или документы OpenAPI. При этом нужно указать, к каким именно объектам и документам применяются изменения (с помощью ссылки на переопределение, состоящей из шаблона пути к файлам и указателя JSON), а также

https://liveinternet.club

имя правила, для которого вы хотите изменить уровень серьезности. Структура переопределений показана на рис. 3.5. Указатель JSON позволяет идентифицировать конкретное значение в документе JSON (с основами синтаксиса таких указателей можно ознакомиться в приложении B).

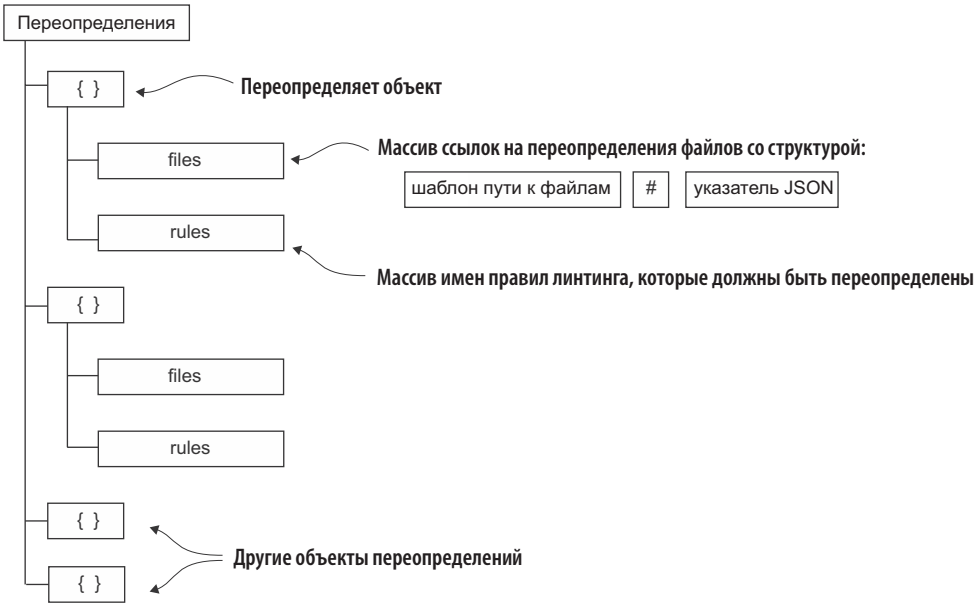


Рис. 3.5. Структура переопределений в Spectral

Посмотрим, как работает переопределение. Нам понадобится файл `chapter3/overrides/openapi.yaml`, часть которого показана ниже. Обратите внимание, что в нем для операции `GET /users/{id}` задан параметр `operationId`, но для операций `PUT /users/{id}` и `DELETE /users/{id}` он отсутствует.

```
openapi: 3.0.0
info:
  title: User API
  version: 1.0.0
servers:
  - url: http://localhost:8080
paths:
  /users/{id}:
    get:
      summary: получает пользователя по id
      operationId: getUserById
      parameters:
        - name: id
          in: path
          required: true
          schema:
```

Параметр operationId определен

https://liveinternet.club

```
          type: integer
      responses:
        '200':
          description: OK
    put:
      summary: обновляет пользователя по id
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: OK
    delete:
      summary: удаляет пользователя по id
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '204':
          description: No Content
```

Параметр operationId не определен

Представьте, что в наборе есть правило `operation-must-have-operationId`, но вы хотите полностью отключить его для всего документа. В таком случае набор правил будет выглядеть как в файле `chapter3/overrides/ruleset1.yaml`:

```
overrides:
  - files:
    - openapi.yaml
    rules:
      operation-must-have-operationId: off
rules:
  operation-must-have-operationId:
    message: ' Каждой операции должен быть присвоен operationId'
    severity: error
    given: '$.paths[*][get,put,post,delete,options,head,patch,trace]'
    then:
      field: operationId
      function: truthy
```

Отключает правило для всего файла

Изменяет серьезность правила на off и тем самым отключает его

Проверьте работу правил на файле `openapi.yaml` из папки `chapter3/overrides`, выполнив команду `spectral lint openapi.yaml --ruleset ruleset1.yaml`. Вы увидите, что в отчете не выявлено нарушений:

```
No results with a severity of 'error' found!
```

Теперь выключите правило линтинга для операции `PUT /users/{id}`, но задайте уровень серьезности `warn`, как показано в файле `chapter3/overrides/ruleset2.yaml`. Имейте в виду, что в указателе JSON на операцию API должно

https://liveinternet.club

использоваться процентное кодирование (то есть `/paths/~1users~1%7Bid%7D/put`, а не `/paths/~1users~1{id}/put`):

```
overrides:
  - files:
      - openapi.yaml#/paths/~1users~1%7Bid%7D/put
    rules:
      operation-must-have-operationId: off
  - files:
      - openapi.yaml#/paths/~1users~1%7Bid%7D/delete
    rules:
      operation-must-have-operationId: warn
rules:
  operation-must-have-operationId:
    message: ' Каждой операции должен быть присвоен operationId'
    severity: error
    given: '$.paths[*][get,put,post,delete,options,head,patch,trace]'
    then:
      field: operationId
      function: truthy
```

Указатель JSON с использованием процентного кодирования

Уровень строгости правила линтинга изменен с error на warn

В результате линтинга этого файла Spectral возвращает только предупреждение для операции `delete`, хотя операция `PUT` содержит ту же проблему:

```
$ spectral lint openapi.yaml --ruleset ruleset2.yaml
36:12 warning operation-must-have-operationId Каждой операции должен быть
присвоен operationId   paths./users/{id}.delete
```

Теперь вы знаете, как переопределять правила линтинга API в Spectral. Если API уже работает в продакшен-среде, при интеграции линтинга разумно использовать постепенное приведение к соответствию с помощью переопределений. Однако не стоит этим злоупотреблять: по возможности лучше стремиться, чтобы API соответствовал руководству по стилю безо всяких исключений.

Как вы могли заметить, при написании правила линтинга важно строго соблюдать его синтаксис. В Spectral необходимо задавать выражение на JSONPath, и для подбора правильного выражения могут потребоваться эксперименты в инструменте для проверки. А что если доверить создание правил искусственному интеллекту (ИИ) и тем самым сэкономить время? Или даже поручить ему генерацию тестов для этих правил, чтобы быстро проверять их корректность? Я покажу, как это сделать, но сперва познакомимся с ИИ-ассистентами.

3.10. ИИ-ассистенты для написания кода

ИИ-ассистенты — это программные агенты, которые выполняют задачи, руководствуясь командами и запросами от пользователя. Благодаря технологиям обработки естественного языка (Natural Language Processing, NLP) они понимают человеческий язык, что позволяет взаимодействовать с ними через простые текстовые промпты. Особое место среди таких помощников занимают ИИ-ассистенты для написания кода, созданные специально для разработчиков

https://liveinternet.club

ПО. Они предлагают и дополняют как строки кода, так и целые функции. В качестве примера можно отметить GitHub Copilot, Tabnine, Amazon CodeWhisperer, Sourcegraph Cody и JetBrains AI.

ИИ-ассистенты помогают разработчикам анализировать и писать код, а также дают актуальные персонализированные рекомендации о том, как его улучшить, например, исправить ошибки. Для этого они передают инструкции и фрагменты кода в большую языковую модель (Large Language Model, LLM) — модель ИИ, обученную на огромных массивах данных, что позволяет ей генерировать ответы, похожие на человеческие. LLM используют методы глубокого обучения, например трансформеры — особый тип нейронных сетей, разработанный для распознавания контекстных связей в языке и решения сложных языковых задач. Благодаря этому модели способны понимать контекст, синтаксис и семантику высказываний. Это, в свою очередь, позволяет им генерировать контент, выполнять переводы и вести беседы с пользователем. Такие модели постоянно развиваются и совершенствуются по мере того, как получают обратную связь и доступ к новым данным.

GitHub Copilot — это ИИ-ассистент, разработанный GitHub с применением LLM от OpenAI. Он поддерживает множество языков программирования и фреймворков и особенно хорошо работает с Python, JavaScript, TypeScript, Go, Ruby, C++ и C#. Copilot автоматизирует рутинное программирование и выступает в роли джуниор-разработчика, тем самым позволяя сосредоточиться на более масштабных задачах, а не тратить время на шаблонный код и борьбу с синтаксисом. Другими словами, Copilot повышает продуктивность разработчиков.

Хотя ассистенты упрощают написание кода, они не освобождают от необходимости изучать основы языка программирования и применяемые инструменты. Разработчики все равно должны проверять и тестировать сгенерированный код. Одна из причин заключается в том, что LLM склонны к «галлюцинациям», то есть могут выдавать неверную, нелогичную или искаженную информацию под видом точных данных. А поскольку LLM не поисковые системы, они обычно не указывают, на чем основан их ответ.

При использовании LLM также стоит помнить о юридических и этических вопросах, связанных с конфиденциальностью данных, лицензированием и авторскими правами. Кроме того, данные для обучения моделей часто собираются из репозиториев, содержащих искажения. А из-за упомянутых галлюцинаций сгенерированный код может содержать логические и синтаксические ошибки или создавать риски для безопасности.

> **СОВЕТ** Если вы хотите узнать больше об LLM и при этом приятно провести время, обратитесь к книге Дэвида Клинтона (David Clinton) «The Complete Obsolete Guide to Generative AI» (Manning, 2023; https://mng.bz/5l2D).

В следующих разделах я расскажу, как генерировать правила линтинга API с помощью Copilot. Все, что для этого нужно, — правильно формулировать промпты для ассистента.

https://liveinternet.club

3.10.1. Промпты в Copilot для создания правил линтинга

Промпт-инжиниринг — процесс составления запросов к ИИ, или промптов, — предполагает передачу LLM инструкций, контекста и примеров, которые позволят ей генерировать более полезные ответы. Расширение GitHub Copilot повышает эффективность промпт-инжиниринга, анализируя контекст кода, а вы, в свою очередь, можете добавить собственные примеры и инструкции, чтобы модель лучше понимала задачу.

Copilot предлагает встроенные подсказки во время набора текста, и вы можете принять их, нажав Tab. В расширении также есть удобный чат для взаимодействия с ассистентом прямо при написании кода. Рассмотрим, как можно использовать этот инструмент для создания правил линтинга.

В разделе 3.6 вы написали правило линтинга, которое проверяет наличие поля `$.info.description` в файле OpenAPI. Теперь сгенерируем это же правило с помощью GitHub Copilot. Сначала создайте файл `chapter3/copilot/rules.yaml`. Откройте его и нажмите ⌘+I в Mac или Ctrl+I в Windows, чтобы вызвать встроенный чат Copilot. В окне чата введите следующий промпт[1]:

```
Напиши правило линтинга для Spectral, которое проверяет
➥ наличие поля info.description в файле OpenAPI
```

Обратите внимание, что LLM не детерминированы, то есть на один и тот же запрос могут генерировать разный код. Я получил следующий результат:

```
rules:
  openapi-info-description-present:
    description: "Проверяет наличие поля info.description field"
    message: "Поле info.description field отсутствует"
    given: "$.info"
    then:
      field: "description"
      function: "truthy"
```

Нажмите кнопку, чтобы принять рекомендованный код. Теперь нужно написать тест, который проверит работу правила. Создайте файл `chapter3/copilot/openapi.yaml`. Откройте его и нажмите ⌘+I в Mac или Ctrl+I в Windows, чтобы вызвать встроенный чат Copilot, и отправьте следующий запрос:

```
Создай документ OpenAPI, чтобы протестировать это правило.
➥ В этом документе правило должно быть нарушено.
```

Я получил от Copilot следующий результат:

```
openapi: 3.0.0
info:
  title: My API
  version: 1.0.0
paths:
```

Свойство info.description отсутствует

[1] Copilot понимает русский язык, но все же рекомендуется писать промпты на английском для получения более точных и стабильных результатов. (Можно переводить на английский с помощью того же ИИ.) В тексте для удобства читателя промпты и ответы модели приведены на русском языке. — *Примеч. ред.*

https://liveinternet.club

```
/path/to/endpoint:
  get:
    summary: Endpoint Summary
    description: Endpoint Description
    responses:
      '200':
        description: Successful response
        content:
          application/json:
            schema:
              type: object
```

Прейдите в папку `chapter3/copilot` и проверьте правило линтинга на тестовом документе OpenAPI, чтобы убедиться, что оно работает:

```
spectral lint openapi.yaml --ruleset rules.yaml
```

Вы должны увидеть сообщение о нарушении правила — это подтвердит, что правило работает корректно.

Вот что нужно знать, чтобы формулировать хорошие промпты для Copilot при работе в VS Code.

- Copilot просматривает файл, над которым вы работаете, и другие открытые файлы в редакторе. Откройте файлы, относящиеся к решаемой задаче, например связанные документы OpenAPI и наборы правил Spectral, — так Copilot получит больше контекста для генерации подходящих рекомендаций.
- Добавьте в начале файла комментарий, поясняющий его содержание, чтобы у ассистента был общий контекст.
- Рекомендую добавить пример кода, который приближен к вашей задаче. Например, включите в набор аналогичное правило линтинга, чтобы дать Copilot понятный ориентир. При желании вы всегда сможете его удалить.
- При работе с кодом (в главе 9 я расскажу о написании правил Spectral и модульных тестов к ним на JavaScript) в начале файла прописывайте `import` или `include`, чтобы подсказать Copilot, какие зависимости, библиотеки и фреймворки вы используете. Также старайтесь комментировать функции и давать им осмысленные имена — это существенно помогает Copilot в генерации кода. В комментариях, помимо прочего, можно подробно описать ожидаемый ввод и вывод функции.
- Продумайте, в каком порядке вы будете писать код. Я предпочитаю сначала составить правило линтинга, а затем попросить Copilot создать документы OpenAPI, на которых можно его протестировать.
- Одной из трудностей при создании правил линтинга в Spectral может стать составление выражений на JSONPath, указателей JSON и регулярных выражений. К счастью, Copilot помогает и в этом. Вы можете попросить его создать нужное выражение и пример кода для его тестирования — со стандартным выводом результата (например, с помощью `console.log()` в JavaScript).

https://liveinternet.club

СОВЕТ Чтобы узнать, как создают регулярные выражения с помощью Copilot, посмотрите видео по ссылке: www.youtube.com/watch?v=dTAZui3k4LU. А из другого видео (www.youtube.com/watch?v=Fi3AJZZregI) вы получите общее представление о возможностях этого инструмента.

Итак, вы узнали, как ассистент помогает создавать, писать и тестировать правила линтинга Spectral с использованием YAML. Опережая вопрос, можно ли работать с правилами линтинга на обычном языке, без необходимости писать код, сразу отвечу да — и расскажу об этом в следующем разделе.

3.11. Автоматический линтинг с помощью LLM

Представьте, что вам нужно создать следующее правило линтинга для всех объектов в запросе и ответе: каждое свойство, имя которого подразумевает денежное значение, обязано хранить числовое значение, а не строку. К примеру, значение свойства `cost_price` (себестоимость) должно быть числом. Конечно, можно создать правило Spectral, которое проверяет только `cost_price`. Но как быть со свойствами `selling_price` (отпускная цена), `vat_amount` (сумма НДС) или просто `price` (цена) и `amount` (сумма)? Список слов, подразумевающих денежные значения, настолько велик, что прописывать их отдельно совершенно непрактично.

Теперь представьте дополнительную задачу: автоматически убедиться, что описание свойства действительно информативно. Скажем, вместо формальной фразы «сумма НДС» для свойства `vat_amount` нужно более развернутое пояснение, например: «Сумма налога на добавленную стоимость (НДС) к цене товара. Зависит от настроенной ставки НДС». Как же написать правило Spectral, которое проверяет информативность описаний? Здесь не поможет ни проверка длины описания, ни регулярные выражения. В Spectral нет простого способа создать такое правило.

Вот еще два универсальных правила линтинга, которые могут оказаться полезными.

- В названиях URI должны быть существительные, обозначающие ресурсы.
- Если операция `GET` принимает параметры, которые, судя по их названию, передают личные данные (например, `date_of_birth` или `first_name`), вместо `GET` необходимо использовать `POST`.

Во многих подобных случаях написать правило Spectral трудно или вовсе невозможно. Обычно такие вхождения отслеживают вручную на этапе ревью API (подробнее о дизайн-ревью API я расскажу в главе 5). Однако существует и автоматизированный способ — использование LLM. Большие языковые модели могут провести линтинг кода по заданным вами правилам и сформировать отчет о нарушениях в указанном формате. Такой подход помогает сократить расходы: анализ API с помощью LLM обойдется дешевле, чем привлечение сеньор-инженера для ручной проверки. К тому же вы сможете формулировать правила на естественном языке, что экономит время. При этом создание правил будет

https://liveinternet.club

доступно всем членам команды, независимо от их опыта работы с синтаксисом Spectral и другими линтерами.

3.11.1. Линтинг с помощью Anthropic Claude

Рассмотрим, как использовать LLM для линтинга OpenAPI. Допустим, вы хотите реализовать правило выше, чтобы свойства, имена которых подразумевают денежные значения, были представлены в числовом формате. Для проверки возьмем документ OpenAPI `chapter3/llm-linting/openapi.yaml`, в котором описана единственная операция: `GET /v1/products`. Она возвращает четыре свойства: `id`, `cost_ price`, `selling_price` и `vat_amount`. Попробуем проанализировать этот документ с помощью LLM Claude от Anthropic (если вы следовали инструкциям в приложении Б, у вас уже должна быть учетная запись).

Авторизуйтесь в сервисе Claude (https://claude.ai). Открыв чат, загрузите в него файл `chapter3/llm-linting/openapi.yaml`. Теперь отправьте следующий промпт:

```
Перечисли все свойства ответа из прикрепленного документа OpenAPI,
➥ имена которых подразумевают денежные значения.
```

В моем случае Claude ответил так:

```
В предоставленном документе OpenAPI к свойствам ответа, имена
➥ которых подразумевают денежные значения, относятся:
- cost_price;
- selling_price;
- vat_amount.
Обоснование:
- cost_price: слово cost ("стоимость") в имени подразумевает
➥ стоимость товара, которая является денежным значением.
- selling_price: слово price ("цена") прямо указывает
➥ на цену продажи товара, что является данными в денежном формате.
- vat_amount: аббревиатура VAT (НДС) означает Value Added Tax ("налог на
добавленную стоимость"), что является
➥ денежной суммой, начисляемой за товары.
Таким образом, имена всех трех свойств явно указывают на то,
➥ что они содержат значения в денежном или валютном формате.
```

Теперь отправьте Claude следующий промпт для линтинга документа OpenAPI:

```
Выполни линтинг документа OpenAPI по следующему правилу:
Если имя свойства подразумевает денежное значение,
➥ это свойство должно быть в числовом формате. Покажи нарушения правила линтинга.
➥ Пример вывода о возможном нарушении: "Свойство "cost_price"
➥ подразумевает денежное значение, но содержит определение "type: string".
➥ Оно должно быть задано с определением "type: number", чтобы
➥ точно представлять денежные данные".
```

В моем случае Claude ответил следующим образом:

```
На основе правила линтинга, которое требует, чтобы свойства, подразумевающие
➥ денежные значения, были указаны в числовом формате, найдены следующие
```

https://liveinternet.club

```
➥ нарушения в документе OpenAPI:
1. Свойство "cost_price" подразумевает денежное значение, но содержит
определение
➥ "type: string". Оно должно быть задано с определением "type: number", чтобы
➥ точно представлять денежные данные.
2. Свойство "selling_price" подразумевает денежное значение, но содержит
➥ определение "type: string". Оно должно быть задано с определением "type:
➥ number", чтобы точно представлять денежные данные.
1. Свойство "vat_amount" подразумевает денежное значение, но содержит
определение
➥ "type: string". Оно должно быть задано с определением "type: number", чтобы
➥  точно представлять денежные данные.
Таким образом, свойства cost_price, selling_price и vat_amount
➥ нарушают правило линтинга, согласно которому свойства, подразумевающие
➥ денежные значения, должны быть в числовом формате вместо строкового. Указание
➥ этих свойств в числовом формате повысит точность API.
```

Хотя вы не передали Claude точные имена свойств, которые нужно проверить, ассистент корректно распознал все четыре свойства и сообщил о нарушении в заданном формате.

Однако интерфейс чата — не самая удобная платформа для специалистов, которые в основном работают в текстовом редакторе или IDE. Поэтому можно подключить такую LLM к линтеру, встроенному в пайплайн CI/CD, с помощью API Optic.

3.11.2. Линтинг с помощью Optic LintGPT

Optic — это автоматизированный инструмент для управления API, который помогает обнаруживать интерфейсы, проверять их на критические изменения, а также сравнивать версии OpenAPI для выявления различий. Он поддерживает ИИ-линтинг с помощью правил, написанных на естественном языке, через LintGPT. Optic является проприетарным инструментом, но предлагает бесплатный тариф. Хотя во время написания книги для использования правил LintGPT требовалось оплатить корпоративную версию, я кратко расскажу, как работает инструмент.

Сначала нужно задать правила LintGPT в файле конфигурации `optic.yaml`, расположенном в корневой папке проекта. Вот пример двух правил LintGPT:

```
ruleset:
  - lintgpt:
      v3_standards:
        rules:
         - "Если имя свойства подразумевает денежное значение,
➥ это свойство должно быть в числовом формате."
        - "Если в ответах на операции GET поле верхнего уровня data содержит
➥ массив, следует использовать разбивку на страницы."
```

Слово «должно» (must) означает, что при несоблюдении правила будет выведена ошибка

Слово «следует» (should) означает, что при несоблюдении правила будет выведено предупреждение

Когда Optic запустит проверку с правилами LintGPT, вы увидите следующие сообщения (Warn — warning, предупреждение; Error — ошибка):

https://liveinternet.club

```
[Warn] В ответе на операцию GET поле верхнего уровня data
➥ содержит массив, но не включает разбивку на страницы.
➥ Разбивка помогает обеспечить производительность и комфортное взаимодействие
➥ с системой при работе с потенциально большими наборами данных.
[Error] Свойства cost_price, selling_price и vat_amount
➥ подразумевают денежные значения, но содержат определение "type: string".
➥ Их следует задать с определением "type: number", чтобы представлять денежные
➥ данные точно.
```

Optic предлагает пользовательский интерфейс, в котором легко определить, где именно в файле определения API возникло нарушение правила. Пример показан на рис. 3.6[1].

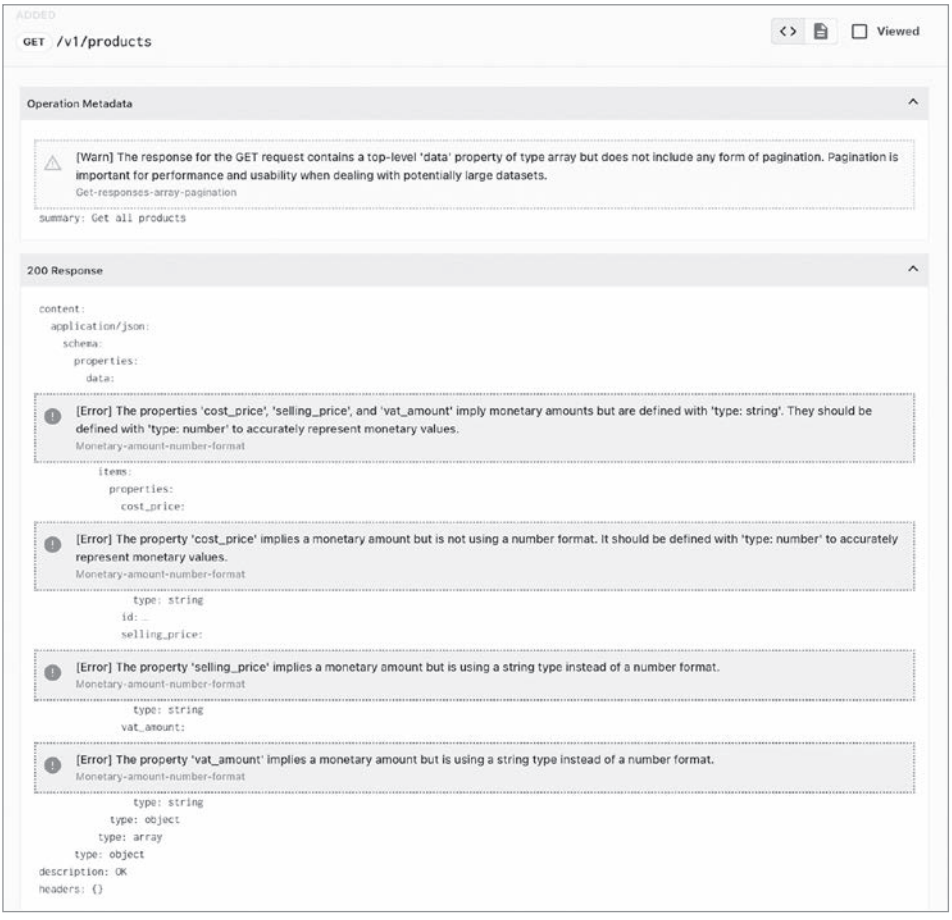


Рис. 3.6. Нарушения правил LintGPT в Optic

[1] Перевод предупреждения и сообщений об ошибках со скриншота см. в тексте выше. — *Примеч. пер.*

https://liveinternet.club

Подведем итоги. Вы узнали, что LLM можно использовать для определения правил линтинга, аналогичных тем, что создаются в традиционных инструментах, например Spectral. Однако LLM позволяют формулировать правила на естественном языке, благодаря чему делать это могут даже те, кто не знаком с синтаксисом конкретных линтеров. Тем не менее при использовании LLM необходимо проверять результаты, чтобы убедиться, что сгенерированные правила действительно помогают выявлять нужные ошибки.

Итоги

- Руководство по стилю API — это справочный документ, содержащий принятые в организации договоренности и рекомендации по проектированию API.
- Линтинг API — это способ автоматизированной проверки того, насколько определение API соответствует руководству по стилю. Он играет важную роль на этапе проектирования в рамках APIOps.
- В качестве инструмента линтинга можно использовать Spectral. Он позволяет применять наборы исполняемых правил, основанных на руководстве по стилю API. Spectral запускается из командной строки, но у него также есть интеграции, плагины и расширения для множества редакторов и инструментов проектирования API.
- При написании правила Spectral вы задаете выражение на JSONPath, которое ищет в документе определения API элементы на проверку, и указываете функцию Spectral для выполнения этой проверки.
- С помощью свойства `extends` можно импортировать наборы правил Spectral — как локальные, так и размещенные удаленно.
- Добавляя новые наборы правил в существующий документ OpenAPI, удобно использовать постепенное приведение к соответствию и отключать проверки выбранных частей файла.
- ИИ-ассистенты для написания кода могут упростить создание правил линтинга API. Кроме того, можно попросить LLM провести линтинг документа OpenAPI по вашим правилам.

https://liveinternet.club

4 Выявление критических изменений: управление развитием API

В этой главе

- ✓ Запуск проверок на наличие критических изменений
- ✓ Использование схем версионирования API
- ✓ Определение политики работы с критическими изменениями

На этапе проектирования продукта API команда разработчиков постоянно совершенствует его и обновляет файлы определения OpenAPI. При этом важно знать, являются ли внедряемые изменения обратно совместимыми. В этой главе я покажу, как с помощью специальных инструментов сравнения выявлять изменения в файлах определения API и находить среди них критические (нарушающие обратную совместимость). Одним из способов избежать критических изменений является версионирование API. Я опишу различные схемы этого механизма и варианты его реализации в организациях, а также объясню, какие затраты связаны с выпуском новых версий API и как донести до потребителей стратегию управления изменениями API с помощью специальной политики работы с критическими изменениями. В завершение я поделюсь советами по управлению критическими изменениями в рабочем процессе. Темы, которые мы затронем в этой главе, будут полезны разработчикам API, менеджерам по продукту API и владельцам таких продуктов.

Для выполнения упражнений вам понадобится инструмент Tufin/oasdiff — если вы следовали инструкциям из раздела Б.8 в приложении Б, он должен быть готов

https://liveinternet.club

к работе. Материалы для упражнений вы найдете в папке `chapter4` репозитория книги (https://github.com/apiopsbook/apiops).

4.1. Изменение определений API в рабочем процессе APIOps

Когда в жизненном цикле продукта API наступит стадия публикации и вы выпустите новый файл определения, потребители API, скорее всего, захотят узнать, что изменилось по сравнению с предыдущей версией. Однако интерес к изменениям может возникнуть еще раньше, на этапе проектирования, и если вас попросят проверить изменения в файле определения API, то важно сначала выявить различия с точки зрения структуры, а не просто сравнить текстовые документы. Например: добавлен ли новый путь? Появился ли в ответе еще один атрибут? Удален ли эндпоинт? Иногда все понятно с первого взгляда — скажем, если изменение небольшое или вы анализируете совершенно новый файл определения API (значит, в нем и все строки новые). Но с крупным файлом определения API, который содержит множество правок, разобраться непросто. Наконец, важно понять, смогут ли потребители, у которых уже настроена интеграция с API, продолжить работу после выпуска обновления, — иными словами, проверить обратную совместимость. На рис. 4.1 отмечены этапы рабочего процесса APIOps, где полезно знать о внесенных изменениях.

4.2. Проблема выявления изменений

Представьте, что вам поручили проверить изменения в файле определения API. Допустим, предыдущая версия хранится в файле `api.yaml`, а новая — в `apiv2.yaml`. В новой версии удален эндпоинт `GET /v1/catalog/categories` и все объекты-ссылки, связанные только с ним. Кроме того, обновлены описание и заголовки эндпоинта `GET /v1/catalog/products` и добавлены два новых заголовка: `x-rate-limit` и `x-rate-limit-remaining`. Изменения показаны на рис. 4.2.

Теперь представьте, что никто не сообщил вам, что именно изменилось. Да, если сравнить текстовые файлы, то вы увидите, какие строки в них различаются. Но как быстро найти операции API, к которым относятся эти строки? Можно открыть файл определения API в визуализаторе и выполнить поиск вручную, но это потребует усилий. Есть ли простой способ быстро увидеть общую картину изменений? В идеале — изменения должны быть видны в структуре документа OpenAPI.

Прежде чем приступить к решению этой задачи, полезно вернуться и вспомнить структуру файла определения OpenAPI. Такие файлы могут быть в формате JSON или YAML, а их корневым объектом является объект OpenAPI. Как и в версии OpenAPI 3.0, он имеет ряд полей (рис. 4.3).

https://liveinternet.club

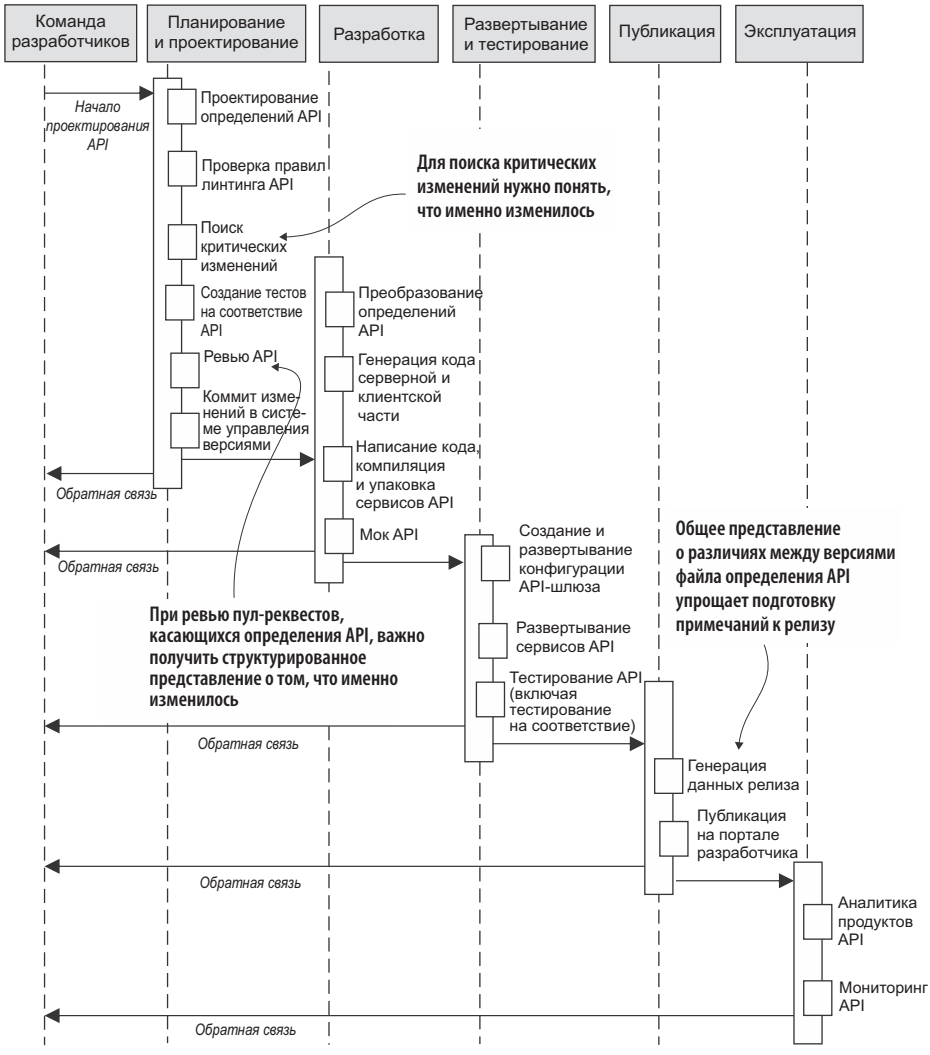


Рис. 4.1. Этапы рабочего процесса APIOps, где важно иметь структурированное представление об изменениях в файлах определения API

Объект OpenAPI `paths` («пути») может содержать несколько объектов `path item` («элемент пути»), каждый из которых, в свою очередь, может включать в себя несколько дочерних объектов операций (рис. 4.4).

Представляя изменения в файле определения API как изменения в структуре документа OpenAPI, вы получаете общую картину. Выявить различия в структуре документа OpenAPI помогут специальные инструменты.

https://liveinternet.club

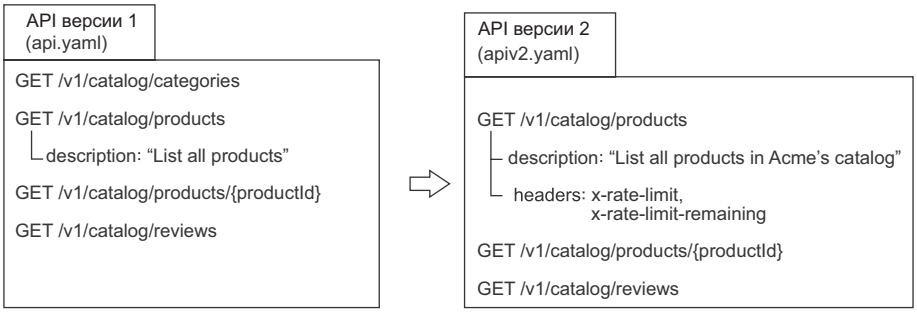


Рис. 4.2. Две версии файла определения API, в одной из которых удален первый эндпоинт и обновлен второй



Рис. 4.3. Структура верхнего уровня документа OpenAPI

https://liveinternet.club



Рис. 4.4. Иерархия объектов OpenAPI: paths, path item и объекты операций

4.3. Выявление различий в API

Любой инструмент для выявления различий в OpenAPI принимает две версии файла определения: «до» и «после» изменений, а затем выводит информацию о том, какие объекты OpenAPI были изменены. В нашем случае, проанализировав файлы `api.yaml` и `apiv2.yaml`, инструмент должен показать, что один эндпоинт был удален, а другой — обновлен. Такой отчет (Diff report) представлен на рис. 4.5.

В этой главе я описываю работу с одним из инструментов для выявления различий — Tufin/oasdiff (https://github.com/Tufin/oasdiff). Однако он не единственный: среди других инструментов с открытым исходным кодом можно также выделить openapi-diff (https://github.com/OpenAPITools/openapi-diff) и optic (https://github.com/opticdev/optic).

> **СОВЕТ** Среди инструментов для отслеживания критических изменений в API существуют также интересные коммерческие решения. Например, Bump.sh (https://bump.sh) предлагает платформу с возможностью проверки на критические изменения в определении API и удобным, автоматически генерируемым журналом изменений. Инструмент Relio (www.getrelio.com), в отличие от других, с которыми я работал, обнаруживает критические изменения, анализируя код приложения. Затем он сравнивает результаты анализа с определениями OpenAPI, чтобы проверить полноту документации и выявить несоответствия.

https://liveinternet.club

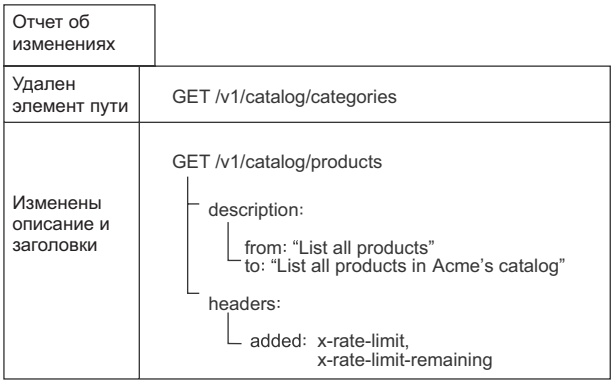


Рис. 4.5. Пример отчета о различиях, в котором показано, что изменилось в новой версии файла определения API

Инструменты для выявления различий в файлах OpenAPI умеют выводить отчеты в разных форматах, включая YAML, JSON, Markdown и HTML. Поскольку они работают через командную строку, их удобно внедрить в пайплайн CI/CD. По умолчанию в Tufin/oasdiff различия выводятся в формате YAML. Я выбрал этот инструмент для книги, поскольку такой вывод соответствует иерархической модели объектов в спецификации OpenAPI (OAS). Обратите внимание, что на момент написания книги в Tufin/oasdiff поддерживается OpenAPI 3.0, но запланирована поддержка версии 3.1.

Откройте проект этой главы и выполните следующую команду, чтобы увидеть разницу между двумя версиями определения API:

```
$ oasdiff diff api.yaml apiv2.yaml
```

Вывод будет таким:

```
paths:
    deleted:
        - /v1/catalog/categories
    modified:
        /v1/catalog/products:
            operations:
                modified:
                    GET:
                        description:
                            from: List all products[1].
                            to: List all products in Acme's catalog[2].
                        responses:
                            modified:
                                "200":
```

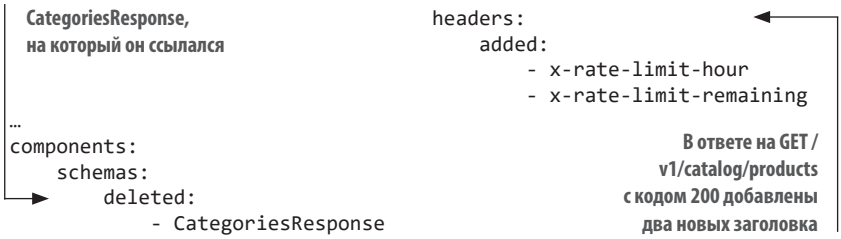
Удален путь /v1/catalog/categories, включая компонент

Описание GET /v1/catalog/products дополнено фразой «in Acme’s catalog»

[1] List all products — Список всех продуктов. — *Примеч. пер.*

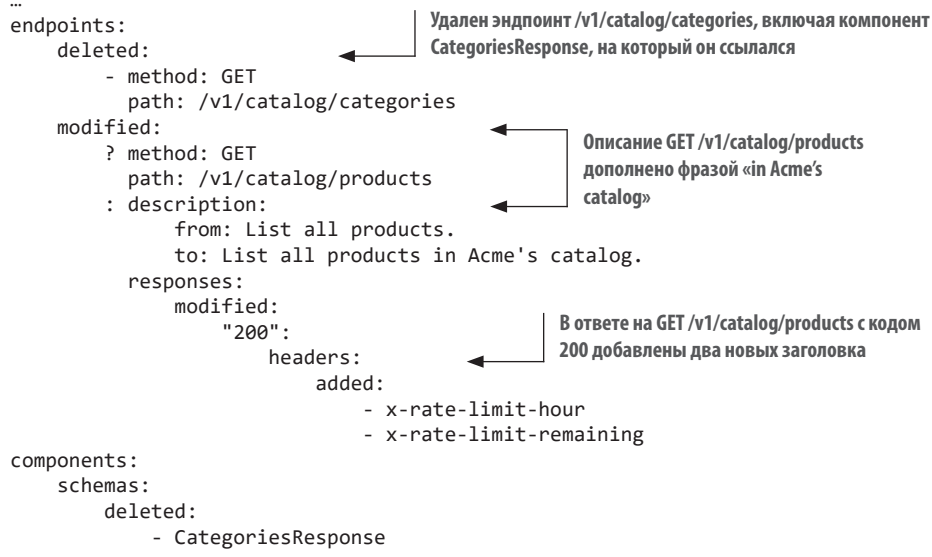
[2] List all products in Acme’s catalog — Список всех продуктов в каталоге Acme. — *Примеч. пер.*

https://liveinternet.club



В отчете о различиях присутствует узел верхнего уровня `paths`, который может содержать три ключа: `added`, `deleted` и `modified`. Каждый из них включает список всех объектов `paths`, которые соответственно были добавлены, удалены или изменены. Если путь указан в категории `modified`, он сопровождается списком измененных объектов и описанием характера изменений, вновь под соответствующими заголовками `added`, `deleted` или `modified`.

Tufin/oasdiff предлагает альтернативную, более простую модель представления различий. Она основана на объектах `endpoints`, а не `paths`. Эндпоинт объединяет путь и операцию, например `GET /v1/catalog/products`. В этой модели узлом верхнего уровня станет `endpoints`, который также может содержать три ключа: `added`, `deleted` и `modified`. Каждый из них включает список всех объектов `endpoints`, которые соответственно были добавлены, удалены или изменены. Аналогично, если эндпоинт указан в категории `modified`, он сопровождается списком измененных объектов и описанием характера изменений, вновь под соответствующими заголовками `added`, `deleted` или `modified`. См. следующую часть отчета:



Единственное отличие между моделями paths и endpoints состоит в способе представления ключей: в модели paths изменение выглядит как узел, чей ключ

https://liveinternet.club

является скалярным значением YAML (строкой пути), тогда как в модели endpoints ключ представлен объектом отображения YAML (комбинацией пути и операции HTTP).

Может ли запись в YAML иметь два элемента в качестве ключа?

Основная структура данных в YAML — узел. Существуют три вида узлов: скалярные (например, строки или числа), узлы последовательности (упорядоченные ряды узлов) и узлы отображения (неупорядоченный набор пар «ключ — значение»). Они показаны на следующем рисунке.

Узлы последовательности начинаются с дефиса (-), а в узлах отображения ключ и значение записываются через двоеточие (:).

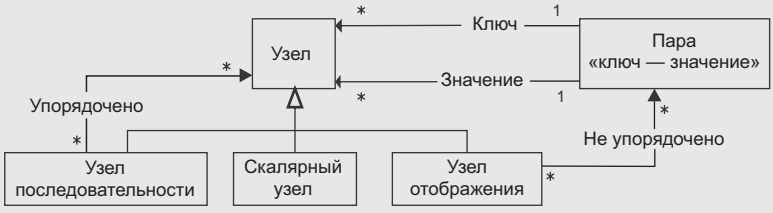


Диаграмма модели узла YAML в формате UML[1]

Вероятно, в большинстве файлов YAML, с которыми вы встречались, в качестве ключей для узлов отображения используются только скалярные узлы. Вот пример узла отображения YAML, где ключ `deleted` выражен скалярным узлом, а значение — узлом последовательности:

```
deleted:
    - method: GET
      path: /v1/catalog/categories
```

Однако в качестве ключа для узла отображения можно также использовать узел последовательности или другой узел отображения. Это называется сложным ключом отображения. Ниже показан пример узла отображения YAML, ключ и значение которого также представлены в виде узла отображения (ключ обозначен символом ?).

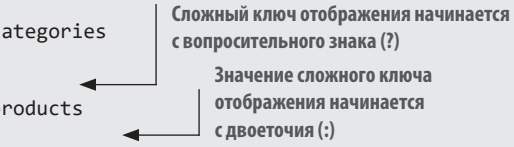
```
? method: GET
  path: /v1/catalog/products
: description:
      from: List all products.
      to: List all products in Acme's catalog.
```

При работе с моделью endpoints инструмент Tufin/oasdiff применяет именно сложные ключи отображения, как показано в примере:

[1] UML — Unified Modeling Language, унифицированный язык моделирования. — *Примеч. пер.*

https://liveinternet.club

```
endpoints:
    deleted:
        - method: GET
          path: /v1/catalog/categories
    modified:
        ? method: GET
          path: /v1/catalog/products
        : description:
              from: List all products.
              to: List all products in Acme's catalog.
          responses:
              modified:
                  "200":
                      headers:
                          added:
                              - x-rate-limit-remaining
                              - x-rate-limit-hour
```



Более подробную информацию о спецификации YAML можно получить на странице https://yaml.org/spec.

Прочитав полный отчет oasdiff о различиях, вы поймете, на чем нужно сосредоточиться при проверке файла `apiv2.yaml`. Если же вместо полного отчета вам нужна лишь краткая сводка изменений, ее можно получить с помощью вот такой команды:

```
oasdiff summary api.yaml apiv2.yaml
```

Результат выполнения команды будет следующим:

```
diff: true
details:
    endpoints:
        deleted: 1
        modified: 1
    paths:
        deleted: 1
        modified: 1
    schemas:
        deleted: 1
```

4.4. Обнаружение критических изменений

Критическое изменение — такое, из-за которого потребитель API вынужден менять код приложения, чтобы оно продолжало функционировать. Иными словами, изменение нарушает обратную совместимость, и клиентские интеграции с API перестают работать полностью или частично. К примеру, если удалить путь (`/v1/catalog/categories` из предыдущего случая), потребители, которые обращаются к этому эндпоинту, начнут получать ошибку 404 Not Found. В отличие от критического изменения, некритическое, или обратно совместимое, не нарушает работу интеграций с API, и потребители могут дорабатывать код в удобное для себя время.

https://liveinternet.club

Представьте, что вам нужно узнать, содержатся ли критические изменения в файле `apiv2.yaml` из проекта этой главы. Вы можете использовать инструмент Tufin/oasdiff и сравнить текущую версию файла с оригинальной, выполнив такую команду в командной строке:

```
oasdiff breaking api.yaml apiv2.yaml
```

Вывод будет следующим:

```
1 breaking changes: 1 error, 0 warning
error [api-path-removed-without-deprecation] at
➥ original_source=api.yaml
   in API GET /v1/catalog/categories
       api path removed without deprecation
```

В результате вы увидите, что удаление пути `/v1/catalog/categories` — это критическое изменение. Однако вывод доступен не только в текстовом формате — вы можете запросить его в виде файла JSON, выполнив следующую команду:

```
oasdiff breaking api.yaml apiv2.yaml --format json
```

В результате вы получите:

```
[
  {
    "id": "api-path-removed-without-deprecation",
    "text": "api path removed without deprecation",
    "level": 3,
    "operation": "GET",
    "operationId": "listCategories",
    "path": "/v1/catalog/categories",
    "source": "api.yaml",
    "section":"paths"
  }
]
```

Отлично! Вы научились автоматически проверять, являются ли изменения в файле определения API критическими. Однако стоит отдельно рассмотреть особый случай критического изменения — завершение поддержки API.

Депрекация (deprecation; также *устаревание*) — уведомление потребителей API о предстоящем удалении интерфейса или его эндпоинта, а *завершение поддержки* (sunsetting) — фактическое прекращение доступа к API на продакшене, которое происходит после закрытия окна депрекации. Мы обсудим эти явления позже.

Теперь представьте, что эндпоинт в файле определения API помечен устаревшим (deprecated) и пришло время его удалить. Скажем, в файле определения `apiv1dep.yaml` из папки `chapter4` эндпоинт `GET /v1/catalog/categories` объявлен как deprecated, а в файле `apiv2.yaml` завершается его поддержка, то есть эндпоинт удаляется. В этом случае вы сталкиваетесь с особым видом критического изменения и, так как уже предупредили о нем пользователей, хотите исключить

https://liveinternet.club

его из автоматизированной проверки. Как игнорировать удаляемый эндпоинт во время проверки на наличие критических изменений?

В инструменте Tufin/oasdiff можно сделать это с помощью расширения OpenAPI под названием x-sunset. Во время проверки oasdiff игнорирует эндпоинты, в определении которых указано поле x-sunset, содержащее прошедшую дату.

Расширения спецификации OpenAPI

Расширения OAS позволяют адаптировать спецификацию для поддержки особых случаев путем добавления данных в файлы определения API. Новые данные можно применять как для улучшения документации, так и для обеспечения корректной работы сторонних инструментов с файлами определения.

Поля свойств, добавленных в расширении, обозначаются префиксом `x-`. К примеру, дата завершения поддержки API будет храниться в поле `x-sunset`. Обратите внимание, что названия, начинающиеся с `x-oai` и `x-oas`, зарезервированы самой спецификацией.

Проверить работу расширения можно, выполнив поиск критических изменений при сравнении файлов `apiv1-dep.yaml` и `apiv2.yaml`. Из следующего фрагмента кода видно, что в файле `apiv1-dep.yaml` эндпоинт `GET /v1/catalog/categories`, помечен устаревшим (`deprecated:true`) и содержит дату завершения поддержки в свойстве `x-sunset`:

```
paths:
  /v1/catalog/categories:
    get:
      deprecated: true
      x-sunset: "2022-08-30"
      tags:
        - Categories
      summary: List all categories[1]
```

Выполните следующую команду, чтобы запустить проверку на наличие критических изменений:

```
oasdiff breaking apiv1-dep.yaml apiv2.yaml
```

Вы не получите сообщения о критическом изменении, поскольку дата 2022-08-30 уже в прошлом. Удалите или закомментируйте строку `x-sunset` и снова выполните проверку. В этот раз вы должны получить сообщение о критическом изменении.

СОВЕТ Посмотреть, какие изменения считаются критическими в Tufin/oasdiff, можно на странице https:// mng.bz/aEAB.

[1] List all categories — Список всех категорий. — *Примеч. пер.*

https://liveinternet.club

Еще одна интересная функция Tufin/oasdiff — возможность перед сравнением изменять префикс эндпоинтов в определении API. Предположим, вы обновляете `api.yaml` до новой версии `apiv3.yaml`, и префикс всех URI изменился с `/v1/` на `/v2/`. Тогда `GET /v1/catalog/products` — то же самое, что и `GET /v2/catalog/products`, изменился только префикс. Игнорируя это различие, вы сможете сосредоточиться на других критических изменениях. Для этого можно унифицировать префиксы URI перед сравнением: oasdiff позволяет указать альтернативный префикс либо для предыдущей версии, либо для новой.

Чтобы убедиться в этом, выполните следующую команду для выявления различий, которая перед сравнением меняет префикс предыдущей версии на `/v2/`:

```
oasdiff diff api.yaml apiv3.yaml --strip-prefix-base /v1/
➥ --prefix-base /v2/
```

Флаг `--strip-prefix-base` удаляет префикс `/v1/` из путей в файле `api.yaml`, а `--prefix-base` добавляет к ним новый префикс `/v2/`. Преобразования происходят только в оперативной памяти и не влияют на файл. Точно так же с помощью флагов `--strip-prefix-revision` и `--prefix-revision` можно изменить префиксы в новой версии.

Мы рассмотрели, как запускать инструменты выявления различий в API и локально проверять наличие критических изменений. В главе 8 я вернусь к этой теме и расскажу о проведении таких проверок в CI-пайплайне. Прежде чем завершить обсуждение инструментов для выявления различий, хотелось бы упомянуть еще одно их применение — генерацию основы для записей в журнале изменений API. Обсудим это в следующем разделе.

4.5. Генерация журнала изменений API

Журнал изменений (changelog) API содержит хронологическую запись всех обновлений интерфейса. В нем должна храниться подробная информация о добавлении, модификации и удалении элементов API. Например, разработчики фиксируют следующее.

- *Эндпоинты*. Список новых эндпоинтов, добавленных для расширения функциональных возможностей, и устаревших, которые подверглись депрекации или были удалены.
- *Параметры*. Список параметров, которые изменили свои типы данных, стали обязательными или необязательными либо были удалены вместе с эндпоинтами.
- *Форматы ответов*. Изменения в структуре данных, возвращаемых API, включая добавление/удаление полей или полную модификацию формата сообщений.
- *Аутентификация и авторизация*. Описание изменений в механизмах безопасности API.

https://liveinternet.club

Просматривая записи в журнале изменений, разработчики узнают об обновлениях API, которые повлияют на их приложения, и могут заранее отредактировать код, чтобы сохранить работоспособность своих продуктов и их совместимость с API. То есть журнал становится каналом связи и помогает потребителям предвидеть изменения и предотвратить проблемы совместимости и стабильности.

Формат и степень детализации журнала неодинаковы у разных поставщиков API. Одни хранят простой перечень изменений, а другие — подробные записи, включающие множество элементов, например:

- информацию о версии и дату ее выпуска;
- описания новых функций, улучшений и исправлений;
- список критических изменений;
- период депрекации;
- информацию о поддержке и ссылки на связанную документацию.

Вот пример записи в журнале изменений API:

```
Журнал изменений API
Версия 2.1.1
Дата: 2023-09-08
Новые функции и улучшения:
• Повышена производительность эндпоинта /search на 20 % за счет механизмов
  кэширования. В эндпоинте /products добавлена поддержка разбивки на страницы
  с использованием параметров limit и offset.
Критические изменения:
• Из эндпоинта /users удален параметр legacy_mode. Вместо него необходимо
  использовать новый заголовок версии.
• Тип данных поля price в ответе от /products изменен со string на float.
Исправление ошибок:
• Устранена проблема, из-за которой пользователи с некоторыми ролями не могли
  выполнить аутентификацию в эндпоинте /admin.
• Исправлено некорректное сообщение об ошибке, получаемое в ответ на
  недействительные запросы в эндпоинте /payments.
Депрекации:
• Флаг old_format в эндпоинте /reports помечен как deprecated. Он будет удален
  в версии 3.0.
Дополнительная информация:
• Если в результате изменений вам понадобится помощь в миграции, свяжитесь
  со службой поддержки.
```

СОВЕТ Другие примеры записей можно найти в журнале изменений API от Intercom (https://mng.bz/gv8E) и Stripe (https://stripe.com/docs/changelog).

Инструменты выявлений различий в OpenAPI сравнивают две версии файла определения API и на основе результатов помогают создать запись в журнале

https://liveinternet.club

изменений. Например, чтобы с помощью Tufin/oasdiff создать краткую запись об изменениях в файле `api2.yaml` по сравнению с `api.yaml`, выполните команду:

```
oasdiff changelog api.yaml apiv2.yaml
```

Вывод будет следующим:

```
2 changes: 1 error, 0 warning, 1 info
error   [api-path-removed-without-deprecation] at api.yaml
        in API GET /v1/catalog/categories
                api path removed without deprecation

info    [api-schema-removed]
        in components/schemas
                removed the schema 'CategoriesResponse'
```

Здесь `error` сигнализирует о том, что изменение критическое. Этот вывод можно использовать как хорошую основу для записи в журнале изменений API, которую (после редактирования, форматирования и дополнения) вы сможете опубликовать на портале разработчиков.

Итак, я рассказал, как выявлять критические изменения и отражать их в журнале изменений API. Далее мы рассмотрим, как такие изменения влияют на развитие API.

4.6. Критические изменения и развитие API

Как я уже говорил, критическим называется изменение, которое нарушает обратную совместимость и вынуждает потребителей API вносить правки в код приложения. В REST API под совместимостью понимается возможность успешного взаимодействия между клиентским приложением и сервером API. Если новая версия API несовместима с клиентским приложением, его приходится адаптировать, иначе обмен данными между клиентом и сервером станет невозможен. Для клиента это оборачивается затратами времени и финансов. Напротив, некритическое (обратно совместимое) изменение не нарушает работу клиента API, и потребители могут дорабатывать клиентскую часть в удобное для них время.

Версия API — совокупность операций API, которые гарантируют долгосрочную совместимость с клиентскими приложениями. В рамках одной версии API предполагается стабильность и отсутствие критических изменений. Если такое изменение потребуется, его обычно вводят в следующей версии. В противном случае обновление придется согласовать с каждым потребителем API, что бывает затруднительно, если API широко используется.

Но зачем поставщику вводить критические изменения? Дело в том, что для развития API необходимо добавлять новые функции и улучшать существующие, а также исправлять ошибки. Некоторые изменения могут оказаться несовместимыми с предыдущими версиями. Однако после публикации API потребители зависят от его реализации. Так что любые шаги для развития интерфейса нужно

https://liveinternet.club

тщательно обдумывать, иначе обновление вызовет сбой в коде клиента. Главная задача — найти баланс между добавлением новых функций, которые улучшают опыт взаимодействия с API, и затратами поставщика на поддержку старых версий. В идеале следует развивать API и одновременно снижать стоимость поддержки старых версий.

> **СОВЕТ** Для небольших организаций, которые только начинают поставлять API, очень важно заранее сообщить внешним потребителям, как планируется его развивать. При публикации первой версии я бы рекомендовал определить основную политику критических изменений и для ее соблюдения внедрить автоматизированную проверку таких изменений в пайплайн определения API (подробнее об этом см. в главе 8).

Мы обсудили, что такое версия API. Теперь посмотрим, как ее можно указывать.

4.7. Схемы версионирования API

Схема версионирования определяет способ, которым потребитель API сообщает поставщику, какую версию API хочет использовать. Для REST API таких схем восемь.

4.7.1. Версионирование на основе пути

Если выбрано версионирование на основе пути (path-based versioning), идентификатор версии указывается в URI запроса API. Обычно шаблон выглядит так: `https://{domain}/{api-name}/{version-identifier}/{operation-id}`. Вот несколько примеров.

- API для проверки статуса платформы Slack предоставляет информацию о ее текущем состоянии, инцидентах, сбоях и техническом обслуживании. API использует семантическое версионирование с указанием идентификатора в параметрах пути:
  - запрос информации об активных инцидентах: `curl https://status.slack.com/api/v2.0.0/current`;
  - запрос истории прошлых инцидентов: `curl https://status.slack.com/api/v2.0.0/history`.
- API службы коротких сообщений от Twilio предназначен для программной отправки SMS. В нем применяется календарное версионирование (CalVer, о котором мы поговорим позже) с указанием идентификатора в параметрах пути:
  - запрос на получение одного сообщения с серверов Twilio: `curl -X GET https://api.twilio.com/2010-04-01/Accounts/{TwilioAccountIdentifier}/Messages/{MessageIdentifier}.json -u {TwilioAccountIdentifier}:{TwilioAuthToken}`.

https://liveinternet.club

- REST API от OpenAPI предназначен для решения задач, связанных с генерацией текста на естественном языке или кода. В нем применяется семантическое версионирование на основе пути, но указывается только основная версия в формате `/v{majorVersion}/`:
  - запрос на перечисление доступных языковых моделей: `curl https://api.openai.com/v1/models -H 'Authorization: Bearer {api-key}`.

Версионирование на основе пути — наиболее распространенная схема. Она понятна, проста в реализации и позволяет клиенту четко указать запрашиваемую версию, а также упрощает навигацию по API через браузер. Однако у нее есть недостаток: так как для каждой версии выделяется новый путь, у поставщика может появиться большое количество API. Технически это нарушает рекомендацию REST, по которой URI должен быть уникальным идентификатором ресурса, — ведь у новой версии API путь меняется, а сам ресурс остается прежним.

4.7.2. Версионирование на основе имени хоста

Версионирование на основе имени хоста (hostname-based versioning) также называется доменным версионированием (domain versioning). Оно схоже с подходом на основе пути, но идентификатор версии указывается не в URI, а напрямую в имени хоста. Обычно шаблон выглядит так: `https://{apiversion}.{domain}.com/uri-path`. Такая схема применяется, например, в TikTok. В API v1 запрос на получение информации о пользователе TikTok выполнялся по запросу: `POST https://open-api.tiktok.com/user/info`. Однако API v2 обслуживается на другом домене, и тот же самый запрос отправляется по новому адресу: `POST https://open.tiktokapis.com/v2/user/info`. При использовании версионирования на основе имени хоста запросы можно направлять на разные серверы, но иногда для работы с новым хостом потребителям API нужно изменить настройки безопасности.

4.7.3. Версионирование на основе строки запроса

В схеме версионирования на основе строки запроса (query string-based versioning) идентификатор версии передается как параметр строки запроса. Обычно шаблон выглядит так: `https://{domain}/{api-name}/{operationid}?{query-string-parameter-versoin-name}={version-identifier}`. Такая структура понятна клиентам, и ее легко использовать. Иногда схема реализована так, что если клиент не указывает версию в строке запроса, то сервер по умолчанию выбирает новейшую, но бывают случаи, когда версия должна быть указана обязательно.

В REST API компании Snyk (https://mng.bz/eo8v) используется версионирование на основе параметра запроса в URI с обязательным указанием идентификатора в формате CalVer. Запрос к API может выглядеть так:

```
curl -X GET "https://api.snyk.io/rest/groups/?version= 2023-03-08~beta
➥ --header "Accept: application/vnd.api+json"
```

https://liveinternet.club

4.7.4. Версионирование на основе пользовательских заголовков

Если выбрано версионирование на основе пользовательских заголовков (custom-header versioning), информация о версии содержится в специальном заголовке, который клиент указывает в запросе. В результате URI ресурса остается неизменным для каждой версии API, что отличает этот подход от версионирования на основе пути. В качестве примера API, использующего эту схему, можно привести REST API от GitHub. Запрос к нему может выглядеть так:

```
curl --header "X-GitHub-Api-Version:2022-11-28" https://api.github.com/zen
```

На момент написания книги, если в запросе не указать номер версии, API GitHub по умолчанию использует версию 2022-11-28.

4.7.5. Версионирование на основе заголовка Accept

Версионирование на основе заголовка Accept (Accept header versioning) также называется версионированием через согласование контента (content-negotiation versioning) или версионированием на основе типа данных (media-type versioning). В нем версия запрашиваемого API указывается в заголовке `Accept` вместе с типом данных. Примеры шаблонов: `Accept: application/vnd.example+ json;version=1.0` и `Accept: application/vnd.example.v1+json`.

Подход применялся в устаревшей версии GitHub REST API V3 — она использовала заголовок `Accept: application/vnd.github.v3+json` (поддержка версии уже завершена). Другим примером служит новый REST API от Tableau. Более подробно я расскажу об этом подходе в разделе 4.8.2.

4.7.6. Версионирование на основе привязки

Версионирование на основе привязки (pinned versioning) также называется версионированием по времени (point-in-time versioning) или динамическим версионированием по дате (dynamic-date versioning). Когда потребитель API впервые отправляет запрос, для его обработки используется текущая версия API, которая «привязывается» к учетной записи. Все следующие запросы будут по умолчанию использовать именно ее. Перейти на новую версию API можно будет вручную — для этого понадобится войти в учетную запись и обновить настройки.

Версионирование на основе привязки используется в API Stripe. Предположим, текущая версия API Stripe имеет номер 2022-11-15, и в своем первом запросе новый потребитель API желает получить список платежей: `curl https://api.stripe.com/v1/charges -u {api-key}`. С момента отправки запроса все последующие обращения к API будут автоматически обрабатываться версией 2022-11-15. Чтобы ее изменить, потребитель должен войти в учетную запись на сайте https://dashboard.stripe.com и в качестве версии по умолчанию выбрать другую. Примечательно, что, хотя в пути Stripe есть параметр версии `/v1/`, он не используется.

https://liveinternet.club

Схожая схема версионирования применяется в Plaid. Следующий запрос к API Plaid позволяет получить информацию о банковском счете, идентификационных номерах и балансе, связанную с учетной записью пользователя в финансовом учреждении:

```
curl --request POST https://sandbox.plaid.com/auth/get
➥ --header 'Content-Type: application/json' --data '{"client_id": {Plaid-
client-id},
  "secret": {Plaid-API-secret},
  "access_token": {access-token}
}'
```

В запросе будет использоваться версия API (например, 2020-09-14), которая привязана к учетной записи пользователя на сайте https://dashboard.plaid.com/team/api.

Еще один поставщик API для оказания финансовых услуг — компания Square (https://mng.bz/ppo8) — также использует версионирование на основе привязки. Stripe, Plaid и Square сочетают эту схему версионирования с другими, чтобы пользователи могли переопределять настройки версии API прямо в запросе. Я расскажу об этом подходе в разделе 4.7.8.

Благодаря версионированию на основе привязки поставщики могут развивать API без изменения эндпоинта и по информации в профилях клиентов отслеживать, какие версии они используют. Однако реализовать такую схему довольно трудно.

4.7.7. Отсутствие схемы версионирования

В некоторых случаях у потребителей API нет возможности указать нужную версию в запросе, хотя она существует. Так, API AWS S3 имеет версию 2006-03-01, но потребитель API не может заменить ее в запросе на другую. Взгляните на пример запроса, который позволяет получить изображение из хранилища S3:

```
curl https://{bucket-name}.s3.amazonaws.com/my-image.jpg
➥ --header "Authorization: {authorization-string}"
```

Как видите, у потребителя нет возможности выбрать другую версию API. Еще один пример — Web API от Slack (https://api.slack.com/web). Он не является REST API в строгом смысле, так как применяет удаленный вызов процедур (Remote Procedure Call, RPC) через HTTP в формате `https://slack.com/api/{method-family}.{methodname}`. Тем не менее его интерфейс удобен для разработчиков, знакомых с REST API. Для поиска сообщений, соответствующих запросу, можно отправить следующую команду:

```
curl https://slack.com/api/search.messages?query=test
➥ --header "Authorization: {authorization-string}"
```

Поставщики API, которые не применяют версионирование, стараются обеспечить стабильную работу своего сервиса и избегают критических изменений.

https://liveinternet.club

Но если такие изменения все-таки потребуются, придется заранее согласовать обновление с потребителями, что бывает весьма затратно.

4.7.8. Комбинация схем

Некоторые API позволяют указывать версию сразу несколькими способами. Ранее мы рассмотрели API Stripe, в котором реализовано версионирование на основе привязки. Но API платформы также поддерживает схему на основе пользовательских заголовков, что позволяет переопределять версию при отправке каждого запроса. Рассмотрим пример:

```
curl https://api.stripe.com/v1/charges -u {api-key}:
➥ -H "Stripe-Version: 2022-11-15"
```

Пользовательский заголовок `Stripe-Version` позволяет указать идентификатор версии в формате даты — введенное значение переопределит стандартные настройки, принятые в учетной записи Stripe. Схожего подхода придерживаются разработчики Plaid (заголовок `Plaid-Version`) и Square (заголовок `Square-Version`).

Стоит отметить, что Stripe дополнительно поддерживает версионирование на основе пути (обратите внимание на параметр `/v1/` в строке запроса). Несмотря на то что разработчики сохранили эту возможность, они вряд ли ею пользуются, поскольку изменение версии существенно повлияет на работу потребителей. Поэтому, как упоминалось выше, в API Stripe версионирование на основе пути фактически не применяется.

В REST API от Foursquare объединены две схемы версионирования: на основе пути и на основе параметра запроса с обязательным идентификатором в формате *календарного версионирования* (Calendar Versioning, CalVer). CalVer представляет собой формат версии API, использующий календарную дату (подробнее в разделе 4.9.2) — она указывается в параметре запроса `'v'`. Допустим, вы используете API Foursquare, чтобы получить список заведений, где подают суши, в определенной локации. В этом случае нужно отправить следующий запрос:

```
curl --request GET
➥ https://api.foursquare.com/v2/venues/search?client_id={client-id}
➥ &client_secret={client-secret}
➥ &ll={latitude,longitued}
➥ &query=sushi&v={YYYYMMDD}
```

Хотя в пути и указан идентификатор версии (параметр `/v2/`), он нужен только для поддержки старых версий и, скорее всего, не изменится.

4.8. Область версионирования API

Мы рассмотрели схемы версионирования API, которые позволяют потребителю запрашивать конкретную версию интерфейса. Но, помимо самой схемы, необходимо учитывать, какую область (уровень) API планируется развивать.

https://liveinternet.club

Например, поставщик может изменить небольшую часть API (один эндпоинт), обновить ресурс, который используется сразу несколькими эндпоинтами, или даже провести масштабные изменения — скажем, внедрить другую систему аутентификации, что повлияет на все операции API. Различные уровни, на которых поставщик может развивать API, называются *областью версионирования*. В этом разделе мы рассмотрим три уровня: глобальное версионирование, версионирование отдельных ресурсов и версионирование эндпоинтов.

4.8.1. Глобальное версионирование

В случае глобального версионирования все операции API имеют одну и ту же версию и обновляются одновременно, чтобы ей соответствовать. Пример – API платформы Zoom, в котором используется глобальное версионирование на основе пути. В первой версии все запросы выполнялись по основному пути `https://api.zoom.us/v1/`, как показано ниже.

- Создать пользователя Zoom:

```
curl --request POST https://api.zoom.us/v1/user/create
➥ -d api_key={api-key} -d api_secret={api-secret}
➥ -d email={email-address} -d type={user-type}
```

- Получить список запланированных встреч пользователя:

```
curl --request POST https://api.zoom.us/v1/meeting/list
➥ -d api_key={api-key} -d api_secret={api-secret}
➥ -d host_id={user_id}
```

- Получить информацию о вебинаре Zoom:

```
curl --request POST https://api.zoom.us/v1/webinar/get
➥ -d api_key={api-key} -d api_secret={api-secret}
➥ -d host_id={user_id} -d id={webinar-id}
```

При обновлении до второй версии API компания Zoom внедрила протокол авторизации OAuth 2.0. Все операции были изменены и стали использовать путь `https://api.zoom.us/v2`, как показано ниже.

- Создать пользователя в Zoom:

```
curl --request POST https://api.zoom.us/v2/users
➥ --header "Authorization: {access-token}"
➥ -d { "action": "create", "user_info": {"email":
➥ "jchill@example.com", "type": 1 } }
```

- Получить список запланированных встреч пользователя:

```
curl --request GET https://api.zoom.us/v2/users/{userId}/meetings
➥ --header "Authorization: {access-token}"
```

- Получить информацию о вебинаре Zoom:

```
curl --request GET https://api.zoom.us/v2/webinars/{webinar-id}
➥ --header "Authorization: {access-token}"
```

Глобальный подход широко используется в случае версионирования на основе пути. Он обеспечивает единообразие всех операций API, а потребителям понятно,

https://liveinternet.club

какая версия используется для обработки запроса. Однако если в API применяется исключительно глобальное версионирование, затраты на обновление могут быть высокими как для поставщика, так и для потребителя, особенно если API содержит много эндпоинтов. Потребителям приходится заново интегрировать функции сервиса в свое приложение.

В результате некоторые поставщики редко обновляют основные версии API. Например, компания Amazon применяет глобальное версионирование в API S3, но не меняет интерфейс уже многие годы. Текущая версия имеет номер 2006-03-01.

4.8.2. Версионирование отдельных ресурсов

При такой схеме версионируется не весь API, а некоторые его ресурсы, то есть вместо глобального обновления поставщик выпускает новую версию конкретного ресурса (или нескольких). Обычно такой подход используется в схемах версионирования на основе заголовка Accept и пользовательских заголовков. Примером может служить API Tableau. В нем новые эндпоинты, реализованные по принципам REST, начинаются с базового пути `/api/-`, а номер версии меняется только в случае критических изменений ресурсов. Рассмотрим пример. Если потребитель API хочет получить список настроек расширения дашборда для сайта, он может отправить следующий запрос:

```
curl --request GET https://us-west-2b.online.tableau.com
➥ /api/-/settings/site/extensions/dasboard
➥ --header "Accept: application
➥ /vnd.tableau.extensions.dashboard.v1.SiteSettings+json"
➥ --header "X-Tableau-Auth: {authentication-token}"
```

Ответ содержит заголовок content-type с версией ресурса:

```
Content-Type:
➥ application/vnd.tableau.extensions.dashboard.v1.SiteSettings+json
{
  "extensions_enabled": true,
  "allow_sandboxed": true,
  ….
}
```

Если в запросе не определен заголовок `Accept`, Tableau по умолчанию возвращает последнюю версию ресурса.

> **СОВЕТ** Больше информации о подходе Tableau к версионированию отдельных ресурсов можно найти на сайте https://mng.bz/OZXR.

Как правило, при версионировании отдельных ресурсов новая версия может оказаться несовместимой с предыдущей. Однако этот подход позволяет лучше контролировать область версионирования. Обновление может затрагивать

https://liveinternet.club

небольшую часть кода. Все операции, которые используют изменяемый ресурс, должны поддерживать новую версию, а остальным клиентам API может потребоваться проверка совместимости — нужно убедиться, что новые изменения не нарушат работу с другими типами ресурсов.

4.8.3. Версионирование отдельных эндпоинтов

При этом подходе каждый эндпоинт (операция) API рассматривается как единица с собственным жизненным циклом выпуска и поддержки, который не зависит от других. Версии одного эндпоинта могут отличаться от версий других эндпоинтов API. Такая схема также называется версионированием по операциям или методам. В качестве примера можно привести REST API Snyk, который использует схему на основе строки запроса с возможностью версионирования отдельных эндпоинтов:

```
GET https://api.snyk.io/rest/groups/{group-id}?version=2023-01-30~beta
GET https://api.snyk.io/rest/groups/{group-id}/settings
➥ /iac?version=2021-12-09
```

Одно из преимуществ такого подхода заключается в точном контроле версий: поставщик может развивать и совершенствовать отдельные эндпоинты, сохраняя поддержку остальных. Поскольку каждый эндпоинт имеет собственный жизненный цикл, его можно обновлять, подвергать депрекации и удалять независимо от других.

Запрашиваемая и фактическая версия

Некоторые API в ответ на запрос пользователя сообщают ему информацию о задействованной версии API или ресурса. К примеру, REST API Snyk использует пользовательские заголовки: `snyk-version-requested` показывает, какую версию эндпоинта запросил клиент, а `snyk-version-served` — какую версию фактически вернул сервер. API Shopify использует похожий подход. Он возвращает пользовательский заголовок `X-Shopify-APIVersion`, чтобы показать, какая версия API была использована для обработки запроса, причем она может отличаться от версии, запрошенной клиентом. Если версии отличаются, это свидетельствует о том, что клиент по-прежнему использует устаревшую и неподдерживаемую версию API и для поддержки актуальной должен обновить код клиентской части.

При версионировании эндпоинтов возникает сложность с созданием единообразного файла определения OpenAPI, так как каждый эндпоинт имеет собственный идентификатор версии и жизненный цикл, что затрудняет взаимодействие между разными их версиями. Чтобы решить эту проблему, компания Snyk размещает файлы определения OpenAPI на портале разработчика, и пользователи могут скачать файлы с заданными идентификаторами версии, например, https://api.snyk.io/rest/openapi/2023-03-03 или https://api.snyk.io/rest/ openapi/2023-02-15~beta.

https://liveinternet.club

Более того, операции в файле определения API содержат дополнительные данные благодаря расширениям спецификации: `x-snyk-api-version` показывает версию API для каждого эндпоинта, а `x-snyk-api-releases` — все версии API, в которых эндпоинт был выпущен.

4.9. Идентификаторы версий API

Помимо схем и областей версионирования, поставщики API могут выбирать, как именно будет выглядеть идентификатор версии. Существуют три основных подхода: SemVer, CalVer и версионирование по уровню стабильности.

4.9.1. SemVer

В спецификации семантического версионирования (Semantic Versioning, SemVer) на сайте https://semver.org рекомендуется указывать идентификатор в формате MAJOR.MINOR.PATCH. Номер MAJOR (старшая версия) увеличивается при выпуске критических изменений, MINOR (младшая) — когда выходят некритические изменения или новые функции, а PATCH (исправление) — при исправлении некритических ошибок.

Если в REST API используют семантическое версионирование, в запросе обычно указывается только номер старшей версии. Например, при версионировании на основе пути старшая версия включается в сам путь (скажем, /v1/ или /v2/). В случае любых критических (обратно несовместимых) изменений старшую версию придется корректировать. Номер младшей версии не указывается. Благодаря такому подходу обеспечивается стабильность API, поскольку изменения добавляются в рамках одной и той же старшей версии.

В некоторых компаниях полная версия формата SemVer размещается в поле `$.info.version` файла определения OpenAPI. К примеру, на момент написания книги в файле определения API Pinterest версии v5 было поле `$.info.version` со значением `5.8.0`, но в URI указывался только номер старшей версии: https://api.pinterest.com/v5/user_account.

> **СОВЕТ** Вы можете найти файл определения OpenAPI Pinterest на сайте https://mng.bz/Y7zB.

Похожий подход используется в API Inventory для eBay, но этот API возвращает еще один заголовок с идентификатором в формате SemVer — `X-EBAY-C-VERSION`.

4.9.2. CalVer

Календарное версионирование (Calendar Versioning, CalVer), упомянутое в предыдущих разделах и подробно описанное на сайте https://calver.org, основано на датах выпуска проекта. В программных проектах используются различные форматы CalVer, но в REST API наиболее распространена схема, в которой идентификатор

https://liveinternet.club

версии совпадает с датой ее выпуска. Так, в API Stripe используется формат ГГГГ-ММ-ДД (например, 2022-11-15), а в API Admin для Shopify применяется укороченный вариант: ГГГГ-ММ. Версию в виде даты проще запомнить, так как мы привыкли ориентироваться по календарю. Как говорилось в разделе 4.7.6, API Stripe совмещает CalVer с версионированием на основе привязки.

4.9.3. Версионирование по уровню стабильности

При выборе этого подхода в качестве идентификатора версии применяются особые обозначения, которые отражают уровень стабильности API, например stable, unstable, beta и alpha. Как уже говорилось, в REST API Admin для Shopify реализовано версионирование на основе пути с идентификаторами в формате CalVer, что видно в следующем запросе: `GET https://{shop}.myshpoify/admin/api/202301/products.json`. Однако Shopify также поддерживает запросы с идентификатором `unstable`, благодаря чему потребители API могут заранее посмотреть разрабатываемые функции и изменения: `GET https://{shop}.myshpoify/admin/api/unstable/products.json`. Версия с идентификатором `unstable` может включать в себя критические изменения, и нет гарантии, что представленные в ней функции действительно будут выпущены.

В некоторых организациях разработчики комбинируют подходы, указывая номер версии в формате CalVer или SemVer вместе с идентификатором стабильности. В REST API для Snyk идентификаторы строятся по шаблону `{calver}~{stability-version}`, например: `2023-03-03~experimental` — экспериментальная версия с нестабильными функциями на стадии разработки, которые могут не дойти до релиза; `2023-03-03~beta` — бета-версия, релиз-кандидат; `2023-03-03` — общедоступная версия, рекомендованная для использования на продакшен.

Версионирование с точки зрения развития продукта API

В докладе API Evolution without Versioning (https://mng.bz/z8eB) Брендан Байерс (Brandon Byars) предлагает не рассматривать версионирование с точки зрения одной лишь архитектуры. Важно помнить, что, если API является продуктом, управление версиями становится частью интерфейса, который вы поставляете пользователям. Иногда поставщики обновляют версию API, чтобы упростить техническую реализацию и избавиться от старого кода, однако такой подход может существенно увеличить затраты потребителей на миграцию. Байерс предлагает три критерия оценки системы версионирования, которую планируется применять в API.

- *Практическая ценность*. Решает ли система версионирования проблемы потребителей API? Уменьшает ли она когнитивную нагрузку и скрывает ли внутреннюю сложность продукта?
- *Удобство использования*. Упрощает ли система версионирования взаимодействие потребителей с API? Предлагает ли она ясное и элегантное техническое решение, которое гарантирует стабильную работу? Система должна быть предсказуемой, интуитивно понятной и легкой в освоении и применении.

https://liveinternet.club

- *Техническая реализация*. Соответствует ли система версионирования принципам качества архитектуры? Делает ли она сервис более понятным и интеллектуально управляемым? Защищает ли система версионирования зависимые подсистемы?

Кроме перечисленных критериев, при работе с продуктом API важно учитывать и другие аспекты версионирования. В своем выступлении The Right Way to Version Your API: The API Consumer's Perspective (www.youtube.com/watch?v=5M8g-4CsOsI&t=1398s) Марьюкка Ниинойя (Marjukka Niinioja) выделяет четыре фактора, которые связаны с управлением продуктом и влияют на стратегию версионирования.

- *Темп развития компании и отрасли*. От некоторых компаний потребители ожидают быстрого развития API, тогда как в других изменения могут быть нежелательны из-за сложностей в миграции. Например, в строго регулируемых секторах и отраслях с большим количеством встраиваемых устройств переход на новую версию API сопряжен с высокими затратами. Это касается, например, сферы здравоохранения и автомобильной промышленности.
- *Взаимоотношения и баланс сил между поставщиком и потребителем*. Более влиятельные поставщики API могут требовать от потребителей перехода на новую версию. Но если влияние и власть на стороне потребителей API, они могут сами определять темп развития сервиса. В отраслях с жестко регламентированными отношениями между поставщиком и потребителем частота обновлений API может быть снижена. Следует также помнить, что критическим считается только то изменение, которое затрагивает используемую функцию; если же она не применяется, обновление не вызовет сбой на стороне потребителя.
- *Поддержка схем версионирования API*. Некоторые старые инструменты управления API и порталы документации могут поддерживать не все схемы версионирования. Важно проверить, работают ли инструменты потребителей API с предложенной схемой.
- *Стратегия управления продуктом API*. По мнению Ниинойи, окончательный выбор стратегии версионирования напрямую связан с тем, насколько хорошо поставщик API понимает ценность своего продукта для рынка. Со временем это понимание может изменить границы API, а также схемы и области его версионирования, чтобы продукт лучше удовлетворял спросу. Вот почему важно начинать работу с такими инструментами, как шаблон ценностного предложения API (описан в главе 5, раздел 5.2.3), — это значительно ускоряет процесс проектирования и помогает быстрее создать схему версионирования, которая учитывает потребности и ожидания потребителей API.

Мы обсудили различные схемы и области версионирования, а также форматы идентификаторов версии. Вы можете комбинировать изученные подходы для управления версионированием API. Далее мы рассмотрим, какие затраты следует учитывать при выпуске новой версии.

4.10. Затраты при переходе на новую версию

Выбор подходящей схемы версионирования и планирование области развития API — важные задачи. Однако не менее важно заранее продумать, как организовать переход потребителей на новые версии API и сообщать им об изменениях. Миграция может сопровождаться существенными затратами.

https://liveinternet.club

При обновлении версии API из-за критических изменений необходимо учитывать определенные расходы. Прежде всего расходы поставщика: на разработку, тестирование, развертывание и поддержку новой версии. Кроме того, ее нужно распространить среди пользователей, дополнив рекомендациями по переходу. Если поставщик API предоставляет доступ к SDK, их также следует обновлять, переиздавать и документировать. При этом приходится поддерживать старые версии API, и тогда возникают альтернативные издержки: ресурсы, затраченные на поддержку старых версий, могли бы пойти на разработку новых функций и совершенствование пользовательского опыта. Именно поэтому поставщики стараются сократить количество старых версий API и побуждают потребителей переходить на новые.

Вторая категория расходов относится к потребителям API. Потребитель вынужден переходить на новую версию, то есть обновлять, тестировать и перевыпускать код клиентской части. И это лишь прямые расходы, но существуют и косвенные. Обновление и выпуск кода клиентской части сопряжены с определенными рисками. При этом также возникают альтернативные издержки: вместо миграции команда могла бы заниматься другими, более важными бизнес-задачами. Даже если поставщик считает затраты на миграцию незначительными, они должны уложиться в бюджет потребителя. В противном случае, когда потребитель не видит существенной выгоды в новой версии API, он просто не захочет на нее переходить.

Жан-Жак Дюбрэ (Jean-Jacques Dubray) описывает три паттерна миграции пользователей на новые версии API (рис. 4.6). Первый из них называется «Узел» (Knot). Согласно ему, все потребители (и окружающая их экосистема) привязаны к определенной версии API и при выпуске новой вынуждены переходить на нее все одновременно. В таком случае, по словам Дюбрэ, с увеличением количества версий совокупные затраты на миграцию за весь жизненный цикл сервиса будут больше, чем при других подходах. Они могут стать настолько высокими, что каждое обновление придется тщательно планировать.

> **СОВЕТ** С подробным математическим анализом расходов на миграцию пользователей API, выполненный Жан-Жаком Дюбрэ, можно ознакомиться на сайте https://mng.bz/0GON.

Второй паттерн называется «Точка-точка» (Point-to-Point) — в этом случае на продакшен одновременно работают разные версии сервиса API. Поскольку не все потребители API готовы сразу выделить время и деньги на миграцию, им предлагают возможность перейти на новую версию в удобный для них срок. Некоторые могут и вовсе не переходить, если текущая версия их устраивает. В отличие от паттерна «Узел», нагрузка ложится на поставщика, так как он должен поддерживать сразу несколько версий API на продакшен. Расходы на миграцию в этом случае ниже, чем при одновременном переходе.

Третий паттерн — «Совместимость». Все потребители обращаются к единому сервису API, который поддерживает совместимые между собой версии. Даже если потребители работают с разными версиями, сервис все равно корректно обрабатывает их запросы.

https://liveinternet.club

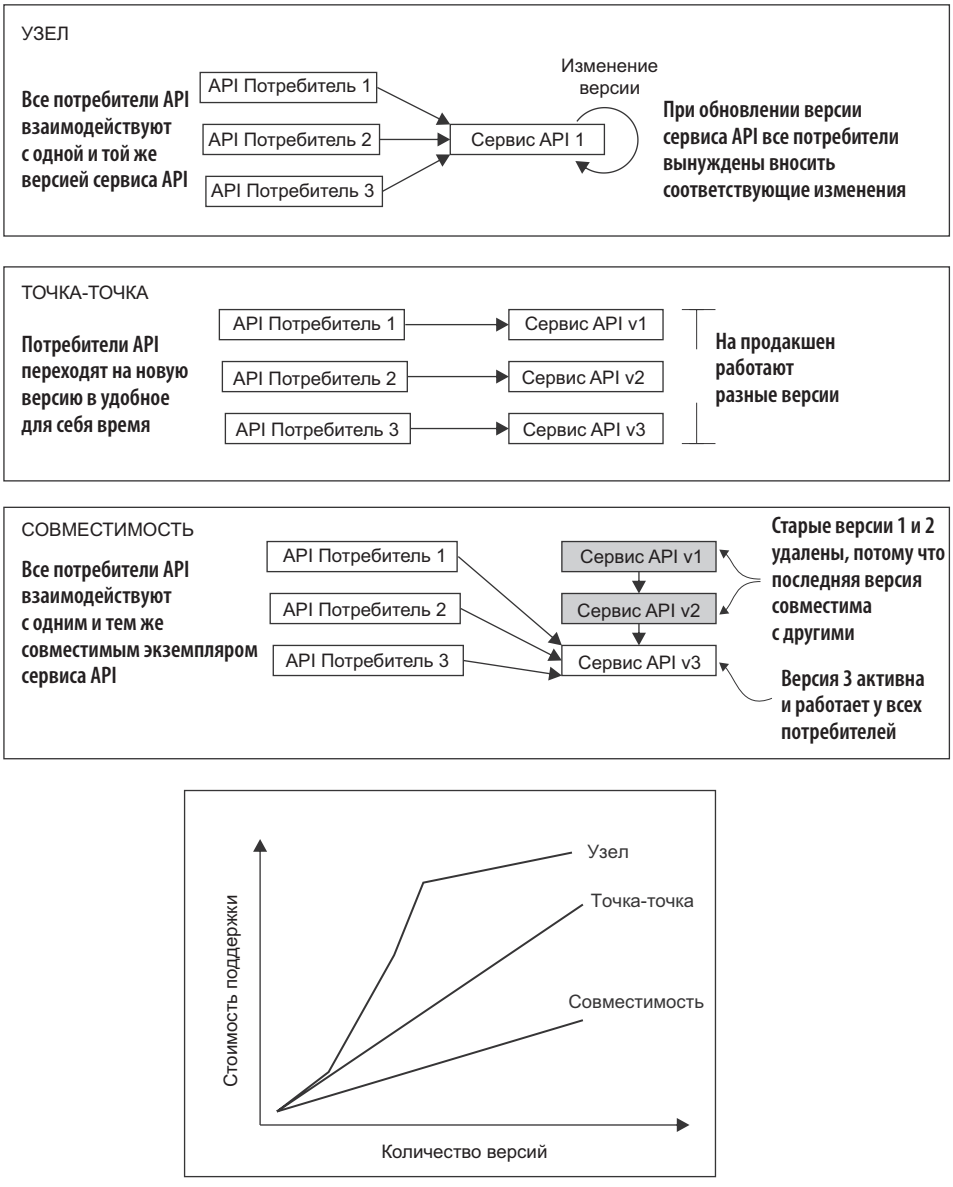


Рис. 4.6. Три модели миграции API по Жан-Жаку Дюбрэ. График показывает затраты, связанные с каждой из них (источник: адаптация графика Дюбрэ с сайта https://mng.bz/0GON, на котором также представлены сопутствующие уравнения)

По оценке Дюбрэ, в долгосрочной перспективе паттерн «Совместимость» наиболее экономичен. Зависимость расходов на миграцию от количества версий по трем моделям также продемонстрирована на графике.

https://liveinternet.club

Примером компании, использующей паттерн «Совместимость», является Stripe. Она внедрила специальный механизм — модуль изменения версии, благодаря которому новейшая версия сервиса API корректно обрабатывает запросы, даже если они обратно несовместимы. Модуль изменения версии представляет собой декларативный код, преобразующий ответ API так, чтобы он соответствовал версии, указанной потребителем. Все модули хранятся отдельно от основного кода сервиса API, который работает на последней версии. Получив ответ от последней версии кода, система применяет модули в обратном порядке: начиная с текущей версии API, они преобразуют данные до тех пор, пока результат не будет соответствовать указанной версии.

Благодаря такому разделению разработчики могут сосредоточиться на основном коде и добавлять новые функции, не отвлекаясь на поддержку старых версий — за преобразование ресурса отвечают отдельные модули. В них содержится описание применяемых трансформаций, а декларативный формат функций позволяет программно генерировать записи в журнале изменений API, чтобы сообщать пользователям об изменениях. Подробнее узнать о подходе Stripe вы можете в блоге компании: https://stripe.com/blog/api-versioning. Еще одна компания, применяющая модель «Совместимость», — Intercom, ее материалы можно найти на странице www.intercom.com/blog/api-versioning.

Модули изменения версий помогают сократить расходы поставщика на внедрение обновления. При этом потребители получают доступ к новым функциям даже через старую версию API. Однако такой подход тоже сопряжен с затратами: на разработку, тестирование и обслуживание кода. В Stripe большинство модулей изменения версии представляют собой чистые функции, преобразующие ответ API. Но некоторые из них выполняют дополнительные действия, например, меняют записи в базе данных. Такие модули сложнее инкапсулировать и поддерживать.

4.11. Политика работы с критическими изменениями

Согласно закону Хайрама (www.hyrumslaw.com), при достаточном количестве пользователей API не имеет значения, что именно вы обещаете в документации: найдется хотя бы один пользователь, который будет опираться на наблюдаемые действия системы. То есть, что бы вы ни указали в определении API, при масштабном использовании сервиса часть потребителей будет полагаться на его фактическое поведение, которое может быть не задокументировано. В результате любое отклонение от текущей реализации может нарушить их работу. Вот почему поставщикам API важно публиковать прозрачную политику работы с критическими изменениями. Это документ, в котором поставщик объясняет, какие изменения считаются критическими, как определяется стабильность API и что следует делать потребителям при выпуске обновлений. Можете считать, что политика работы с критическими изменениями — ваш свод правил по управлению развитием API.

https://liveinternet.club

Политику работы с критическими изменениями следует публиковать на портале разработчика, чтобы она была доступна всем потребителям API. В некоторых организациях предпочитают называть этот документ политикой версионирования. Однако каким бы ни было его название, документ должен содержать следующую информацию:

- схему версионирования, область версионирования и идентификаторы версии API;
- список изменений API, которые считаются критическими и некритическими;
- уровни стабильности API;
- план выпуска версий API.

Взгляните на примеры политик работы с критическими изменениями в REST API.

- Политика версионирования, поддержки и работы с критическими изменениями в Microsoft Graph (https://mng.bz/KZnj).
- Политика работы с критическими изменениями API LinkedIn (https://mng.bz/5lza).
- Критические изменения на платформе HubSpot (https://mng.bz/9d87).
- Политика версионирования API Onfido (https://mng.bz/67By).

Схемы версионирования, область версионирования и формат идентификаторов версий мы уже рассмотрели в разделах 4.7–4.9. Далее подробно остановимся на трех других темах: критических изменениях, уровнях стабильности и плане выпуска версий API.

4.11.1. Определение критических и некритических изменений

Чтобы потребители API могли разрабатывать клиентскую часть интеграции, важно заранее определить и опубликовать список изменений, которые вы считаете критическими и некритическими. Впоследствии он станет основой для автоматизированных проверок на критические изменения с помощью инструментов для выявления различий в OpenAPI или специальных скриптов. Примеры критических изменений приведены в табл. 4.1, однако учтите, что каждая организация по-своему трактует, что к ним относится. Например, я считаю добавление нового значения enum (перечисляемого типа) критическим изменением, но другие организации, включая Onfido и GitHub, придерживаются иной точки зрения. (Вопросы, связанные с добавлением новых значений enum, рассматриваются в разделе 4.12.) Кроме того, сами инструменты выявления различий в API по-разному оценивают изменения.

Именно по этим причинам важно публиковать собственный список. Благодаря ему вы сможете не только направлять ожидания потребителей API, но и проверять, работает ли выбранный инструмент выявления различий в OpenAPI

https://liveinternet.club

в соответствии с вашей логикой. В противном случае его, возможно, понадобится дополнить собственными скриптами.

Таблица 4.1. Примеры критических изменений в файле определения OpenAPI

Объект API	Категория изменения	Изменение
Путь	Удаление	Удаление эндпоинта
	Добавление	Добавление нового обязательного параметра в пути
Запрос	Добавление/увеличение	Добавление нового обязательного свойства в тело запроса
		Добавление нового шаблона проверки данных в схему запроса
		Добавление нового обязательного параметра заголовка
		Увеличение минимального количества элементов массива в запросе
	Изменение	Объявление необязательного параметра в заголовке запроса обязательным
		Объявление необязательного свойства в теле запроса обязательным
		Изменение проверенного шаблона в параметре запроса
	Удаление/уменьшение	Удаление значения enum
		Сокращение максимального количества элементов массива в запросе
		Сокращение максимальной длины строки
Ответ	Добавление/увеличение	Увеличение максимальной длины массива в ответе
		Добавление нового значения enum (совместимость изменения зависит от кода клиентской части)
	Изменение	Объявление обязательного свойства в теле ответа необязательным
		Объявление обязательного заголовка ответа необязательным
	Удаление/уменьшение	Удаление обязательного свойства
		Удаление типа данных из ответа
		Удаление значения enum
		Удаление обязательного заголовка ответа
		Сокращение минимального количества элементов массива в ответе
		Сокращение минимальной длины строки в ответе

В этой таблице содержатся только критические изменения, а список некритических вы можете составить самостоятельно. Обычно к ним относят различные добавления, например добавление эндпоинтов, необязательных параметров и заголовков в запросе или новых полей и заголовков в ответе.

4.11.2. Стабильность версии API

Стабильность API — гарантии, которые дает поставщик в отношении того, как будет меняться версия интерфейса. Если указать уровень стабильности,

https://liveinternet.club

потребители API будут знать, как часто им придется обновлять свой код после интеграции. Вы, в свою очередь, сможете получать обратную связь от потребителей касательно предстоящих изменений по мере развития API.

У каждого API должны быть определены гарантии стабильности. Так, Google предлагает разработчикам следующее руководство по уровням стабильности своих API, изложенное в документе AIP-181 (https://google.aip.dev/181).

- *Alpha*. Команда активно дорабатывает API, возможны критические изменения. Версию использует ограниченный круг пользователей, для которых приемлемы частые обновления.
- *Beta*. Разработка API завершена, и после открытого тестирования версию планируют объявить стабильной. Она может быть открыта широкому кругу пользователей, что фактически означает ее публичное размещение. Команда по-прежнему может вносить критические изменения, но они должны вступать в действие после определенного периода депрекации, чтобы пользователи успели завершить миграцию кода. При этом рекомендуется стремиться к максимальной стабильности бета-версии API.
- *Stable*. Стабильная версия API должна полностью поддерживаться и быть обратно совместимой. Критические изменения допускаются только в случаях, связанных с безопасностью или требованиями законодательства.

REST API платформы Ghost (https://ghost.org/docs/content-api) определяет другие уровни стабильности.

- *Experimental*. Экспериментальная версия API, над которой активно работают и которая допускает критические изменения.
- *Stable*. Стабильная версия, в которой возможны только некритические (обратно совместимые) изменения.
- *Deprecated*. Устаревшая версия, которую планируется удалить.

Независимо от того, какую систему уровней стабильности вы выберете, важно четко задокументировать ее на портале разработчика, чтобы потребители API могли легко в ней разобраться.

4.11.3. График выпуска версий API

Помимо прочего, важно заранее документировать, как долго вы планируете поддерживать стабильную версию API, — это позволит потребителям планировать обновление. К примеру, Shopify выпускает новую стабильную версию API каждые 3 месяца и поддерживает ее в течение 12 месяцев. Благодаря такому подходу у потребителей API есть минимум 9 месяцев для тестирования своих приложений и их переноса на следующую стабильную версию. Предсказуемый цикл позволяет им планировать свою работу.

https://liveinternet.club

СОВЕТ Подробную информацию о графике выпуска API Shopify вы найдете на странице https://mng.bz/oe6M.

При описании уровней стабильности API следует также указать, как вы проводите депрекацию и завершаете поддержку API, поскольку это неотъемлемая часть его жизненного цикла. Объявление о депрекации — официальное уведомление о том, что HTTP-ресурс устарел и его больше не рекомендуется использовать. Вы сообщаете потребителям, что перестанете поддерживать API или закроете к нему доступ. С этого момента им следует готовиться к миграции на другую версию и избегать разработки новых приложений, зависящих от устаревшего API. Пометка устаревшего API обозначается в OAS с помощью специального поля Deprecated, которое принимает значение типа boolean. Его можно задать для операций, параметров и объектов схемы OpenAPI.

Когда истекает период депрекации, поддержка API завершается. В день завершения поддержки команда удаляет API с сервера и закрывает к нему доступ, а при попытке обращения к его эндпоинту потребители получают ошибку 404 Not Found. Рисунок 4.7 демонстрирует разницу между датами объявления API устаревшим и завершения поддержки.

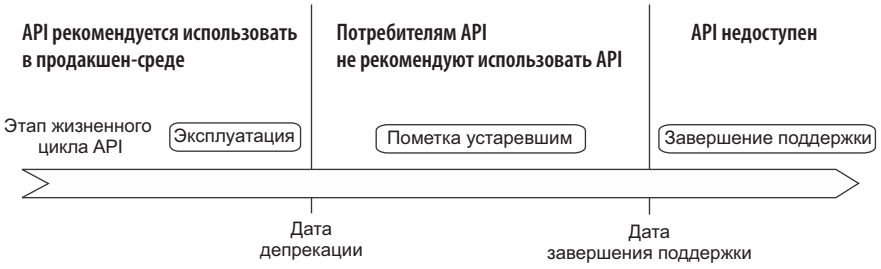


Рис. 4.7. Даты депрекации и завершения поддержки API. После объявления депрекации использовать API не рекомендуется. Со дня завершения поддержки он становится недоступен

Завершение поддержки API — критическое изменение, ведь потребители больше не смогут использовать сервис. Однако если заранее объявить депрекацию и дать им достаточно времени на миграцию, то в рамках автоматизированных проверок можно не считать отключение API критическим изменением. Если вы автоматизировали выявление таких изменений в пайплайне CI/CD, можно настроить его так, чтобы запланированное завершение поддержки не вызывало ошибку.

4.12. Управление критическими изменениями

Как мы уже обсуждали, при версионировании API необходимо учесть множество аспектов: выбор схемы и области, определение уровней стабильности,

https://liveinternet.club

взаимодействие с потребителями и планирование обновлений. Согласно закону Хайрама, критическим может стать любое изменение — как в дизайне, описанном в контракте API, так и в наблюдаемом поведении системы, на которое полагаются потребители. Выявлять критические изменения на уровне дизайна помогает такой инструмент сравнения версий, как Tufin/oasdiff. Но в зависимости от кода и рабочего процесса могут понадобиться и другие средства управления. Вот пара рекомендаций о том, как управлять критическими изменениями на практике.

Продумайте систему версионирования

Разрабатывая новый API, определите возможную схему версионирования, даже если пока ее не используете. Вы сможете обратиться к ней в будущем, если возникнет необходимость. Подберите такую схему (или комбинацию схем), которая поможет развивать требуемые области API. Кроме того, выберите формат идентификатора — он должен логично соответствовать вашему проекту и быть понятным для потребителей API. Благодаря продуманному версионированию будет проще описывать изменения в журнале и документации API.

Обеспечьте расширяемость дизайна

Следуйте принципу открытости/закрытости Бертрана Мейера (Bertrand Meyer), который гласит, что программное обеспечение должно быть открыто для расширения, но закрыто для изменения. Применительно к API это означает, что ресурсы следует проектировать таким образом, чтобы к ним можно было добавлять новые свойства, но без возможности менять существующие, от которых зависит работа потребителей.

Если набор возможных значений еще не определен или вы предполагаете, что он может измениться, используйте открытый список значений вместо перечисления. Перечисление (enum) — это свойство с фиксированным набором значений. Код клиентской интеграции — будь то написанный вручную или автоматически сгенерированный на основе определения OpenAPI — может некорректно обрабатывать новые или неизвестные значения. Если их добавить в API, возможно, изменение окажется критическим и код выдаст ошибку. Поэтому перечисления рекомендуется применять только там, где вы уверены, что список значений окончательный. Однако, учитывая постоянное развитие API, на ранних этапах проектирования, когда требования еще не до конца понятны, трудно гарантировать неизменность списка. Поэтому, если вы допускаете, что со временем он может дополняться, лучше сразу выбрать открытый список значений. Например, постарайтесь избежать следующего перечисления:

```
properties:
  role:
    type: string
    enum:
    - all
    - admin
    - member
```

https://liveinternet.club

Замените его открытым списком:

```
properties:
  role:
    type: string
```

Специалисты Zalando рекомендуют сообщать потребителям API текущий список поддерживаемых значений с помощью расширения `x-extensible-enum`. Вот как его можно применить:

```
properties:
  role:
    type: string
    x-extensible-enum:
    - all
    - admin
    - member
```

Но в таком случае потребители API должны быть готовы к расширению перечисления и реализовать логику обработки неизвестных значений (https://mng.bz/ng5K). В то же время поставщики должны с осторожностью добавлять новые элементы в перечисление — так, чтобы не нарушить семантику его существующих значений.

Если вы уверены, что перечень значений в определении API больше не изменится, то можете преобразовать список строк в перечисление, поскольку такое изменение не будет считаться критическим. Благодаря тому, что перечисление предполагает строгую проверку типа, новые клиентские интеграции получат более надежную защиту от ошибок.

Еще один принцип проектирования расширяемых API — не возвращать массивы JSON в качестве корневого элемента ответа. Под корневым массивом JSON понимается документ, у которого верхним элементом является не объект, а массив JSON, например:

```
[
  {
    "name": "Acme Rope Toy",
    "description": "Acme Rope Toy подарит веселье и радость вашей собаке",
    "price": 100
  },
  {
    "name": "Acme Dog Food",
    "description": "Полезная еда для вашей собаки",
    "price": 50
  }
]
```

В такой JSON-документ добавление нового атрибута верхнего уровня с дополнительной информацией (скажем, объект meta или pagination) будет критическим изменением. Но этого изменения можно избежать, если обернуть массив в корневой объект JSON:

https://liveinternet.club

```
{
  "data": [
    {
      "name": "Acme Rope Toy",
      "description": "Acme Rope Toy подарит веселье и радость вашей собаке",
      "price": 100
    },
    {
      "name": "Acme Dog Food",
      "description": "Полезная еда для вашей собаки",
      "price": 50
    }
  ],
  "pagination": {
    "self": "https://api.acme-pet-supplies.com/v1/catalog/products?
➥ cursor=bccf5512",
    "prev": "https://api.acme-pet-supplies.com/v1/catalog/products?
➥ cursor=0242ac120",
    "next": "https://api.acme-pet-supplies.com/v1/catalog/products?
➥ cursor=651b1f06"
  }
}
```

Поэтому в качестве корневой структуры данных в запросах и ответах API рекомендуется всегда использовать объект, а не массив JSON.

Применяя открытые списки (такие, как `x-extensible-enum`) и избегая корневых массивов JSON, вы сможете проектировать расширяемые API с наименьшим риском критических изменений. Кроме того, полезно проводить ревью API. Простые, неформальные проверки также помогают выявить возможности для расширяемости API. О ревью API мы поговорим в главе 5.

Итоги

- Благодаря структурированному списку различий между версиями файла определения API проще отслеживать изменения в дизайне OpenAPI.
- Критические изменения вынуждают потребителей API обновлять клиентские приложения. Такие изменения можно автоматически выявлять с помощью инструментов для сравнения версий OpenAPI.
- Версионирование API — способ развивать интерфейс, не нарушая интеграцию с существующими клиентами. Есть несколько подходов к управлению версиями, включая различные схемы, объемы и форматы идентификаторов, что позволяет выбрать именно ту модель, которая лучше всего подходит для вашего API. Тем не менее версионирование API следует считать частью поставляемого интерфейса и не забывать о его удобстве для пользователей.
- Политика работы с критическими изменениями должна содержать список изменений, которые вы считаете критическими и некритическими, описание уровней стабильности API и график выпуска версий. Политику необходимо опубликовать, чтобы потребители API могли учитывать ее при планировании своих интеграций.

https://liveinternet.club

5 Дизайн-ревью API: проверка компонентов, недоступных для автоматизации

В этой главе

- ✓ Что такое ревью API
- ✓ Процесс ревью
- ✓ Оценка эффективности ревью

Предположим, вы разработали новый API, и его файл определения прошел все автоматизированные проверки на этапе коммита изменений в рамках пайплайна непрерывной интеграции и непрерывной доставки (CI/CD). Вы создаете пул-реквест, чтобы слить изменение с основной веткой проекта. Возникает вопрос: должны ли ваши коллеги дополнительно проверить эти изменения? Ответ — да. Ревью API совместно с коллегами или даже потенциальными пользователями дает обратную связь, которую невозможно получить с помощью инструментов автоматизации. Кроме того, появляется отличный повод внимательно изучить отчеты автопроверок. В этой главе я расскажу, как встроить ревью API в рабочий цикл APIOps и получать отзывы коллег. Хотя информация будет полезна всем, кто участвует в разработке API, знания о том, как выстроить эффективный процесс ревью, особенно пригодятся владельцам API и специалистам, отвечающим за его единообразие.

https://liveinternet.club

5.1. Что такое дизайн-ревью API

Ревью API — способ получить обратную связь от коллег и внутренних потребителей, чтобы усовершенствовать дизайн API и узнать, соответствует ли он стандартам организации в отношении единообразия и удобства использования. В рамках APIOps ревью инициируется на этапе проектирования — при создании пул-реквеста на внесение изменений в файл определения API (рис. 5.1). Оно должно проводиться после автоматического линтинга и выявления критических изменений. Это отличает ревью от традиционной проверки на соответствие руководству по стилю, которая целиком выполняется вручную и часто приводит к ошибкам, связанным с человеческим фактором. Ревью API дополняет автоматические проверки ручными, позволяя проверить то, что невозможно автоматизировать. Например: охватывает ли дизайн API все необходимые операции? Согласна ли команда с тем, как называются ресурсы и атрибуты? Ревью позволяет обнаружить невыявленные ранее проблемы единообразия, которые впоследствии можно предотвратить с помощью автоматизации, используя полученную информацию для настройки проверок.

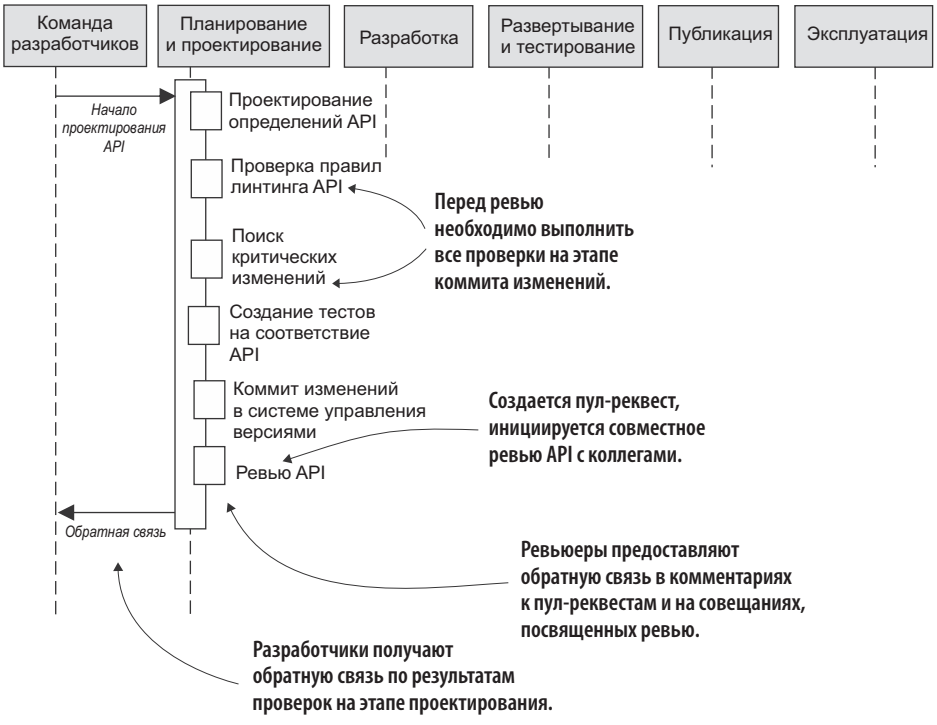


Рис. 5.1. В рабочем процессе APIOps ревью API выполняется на этапе проектирования: начинается после успешного прохождения автоматизированных проверок и завершается после устранения всех замечаний ревьюеров и слияния изменений с основной веткой

https://liveinternet.club

Ревью API можно проводить очно или асинхронно. При асинхронном подходе разработчик API создает для коллег пул-реквест на изменения в определение API. Ревьюеры изучают изменения и сопроводительные метаданные, визуализируют структуру файла определения, чтобы понять, как будет выглядеть API, а также выполняют пробные запросы к автоматически сгенерированному моку API, чтобы проверить, как интерфейс будет вести себя на практике. Ревьюеры могут оставлять комментарии прямо в системе контроля версий с помощью таких инструментов, как GitHub, GitLab или Bitbucket, в удобное для себя время. Разработчик читает их и при необходимости меняет API. Такой упрощенный процесс ревью и утверждения рекомендуется, если изменения незначительны или строго соответствуют руководству по стилю, принятому в организации.

Однако бывают случаи, когда для обсуждения требуется личная встреча — так называемое очное дизайн-ревью. Необходимость в нем возникает, когда пул-реквест затрагивает значительный объем API или когда разработчик должен объяснить обширный контекст, чтобы рецензенты поняли суть изменений. Кроме того, очный формат предпочтителен, если проект предложила менее опытная команда, которой еще предстоит освоить подход design-first, развить продуктовое мышление (ориентированное на пользователя) или научиться применять в разработке API моделирование предметной области. Во время очного ревью рецензенты приглашают разработчиков, чтобы обсудить пул-реквест, понять причины изменений, услышать обоснование проектных решений, оценить единообразие API, а также предложить поддержку и конкретные рекомендации для совершенствования API. В некоторых организациях практикуют комбинированный подход: начинают с асинхронного ревью, а при необходимости — например, в случае крупных изменений или решений, требующих обстоятельного обсуждения, — организуют очную встречу. Если организация небольшая, ревью может проводиться неформально, с участием только одного коллеги, но в крупных компаниях для этого иногда выделяют специальную команду управления API. Вновь подчеркну: независимо от формата — очного или асинхронного — ревью следует проводить только после завершения всех автоматизированных проверок.

Если ревьюеры согласны с предложенными изменениями, они утверждают пул-реквест. Тем самым фиксируется контракт API, на основе которого могут работать команды серверной и клиентской части. Это позволяет вести параллельную разработку и ускорить доставку API.

Однако обратная связь от ревьюеров должна поступать не только после отправки пул-реквеста, но и на более ранних этапах проектирования API. В зависимости от сложности предметной области разработчики API могут подготовить модель бизнес-операций и структуры данных на основе user stories (пользовательских историй), job stories (рабочих историй, о них расскажу ниже) и модели предметной области. Ревьюеры помогают разработчикам анализировать модель API, чтобы убедиться, что она соответствует ограниченному контексту, а также принципу единственной ответственности, инкапсуляции и другим концепциям проектирования.

https://liveinternet.club

Предметно-ориентированное проектирование

Модель предметной области — это структурированная картина бизнес-концепций, отношений между ними и рабочих процессов, представленная в виде взаимосвязанных программных объектов с определенными данными и поведением. Предметно-ориентированное проектирование (Domain-driven Design, DDD) распространилось благодаря книге Эрика Эванса (Eric Evans) «Domain-Driven Design: Tackling Complexity in the Heart of Software»[1] (Addison-Wesley, 2003). Это метод проектирования ПО, при котором тщательно анализируют бизнес-процесс и по результатам анализа создают модель предметной области. Разработчики ПО продумывают единый язык — общий, повсеместно используемый и строго определенный инструмент коммуникации, который позволяет однозначно и единообразно обсуждать концепции предметной области как со специалистами, так и с пользователями. С его помощью разработчики внедряют термины предметной области в создаваемую программную систему.

Один из подходов DDD предполагает выделять *ограниченные контексты* (bounded context): в крупных бизнес-системах модель предметной области разбивается на группы с четко определенной взаимосвязью. Внутри ограниченного контекста термины и концепции предметной области используются единообразно.

Описав модель и операции API, разработчики могут преобразовать их в дизайн — внести изменения в файл определения OpenAPI. При этом они опираются на руководство по стилю API, принятое в организации, чтобы созданные и обновленные файлы определения соответствовали внутренним стандартам. Дополнительно разработчики подключают (напрямую или с помощью ссылки) общие, повторно используемые библиотеки компонентов OpenAPI, которые требуются в проекте. Это избавляет от необходимости писать код с нуля и упрощает процесс ревью за счет применения уже одобренных компонентов. Разработчик отправляет пул-реквест, который проходит серию автоматизированных проверок. Если в процессе линтинга и проверок на наличие критических изменений обнаруживаются ошибки, необходимо сначала устранить их и только потом приступать к ревью API. Когда пул-реквест готов к ручной проверке, ревьюеры просматривают его, одобряют, отклоняют или откладывают до дальнейших очных обсуждений. Подход проиллюстрирован на рис. 5.2.

Кто при проверке пул-реквеста инициирует очное дизайн-ревью? Такое решение остается на усмотрение ревьюеров. Обычно они приглашают разработчиков, если внедряются новые паттерны API или изменения в запросе существенно отклоняются от руководства по стилю API и требуют более обстоятельного обсуждения, которое сложно провести в комментариях.

[1] Э. Эванс. «Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем». — *Примеч. пер.*

https://liveinternet.club

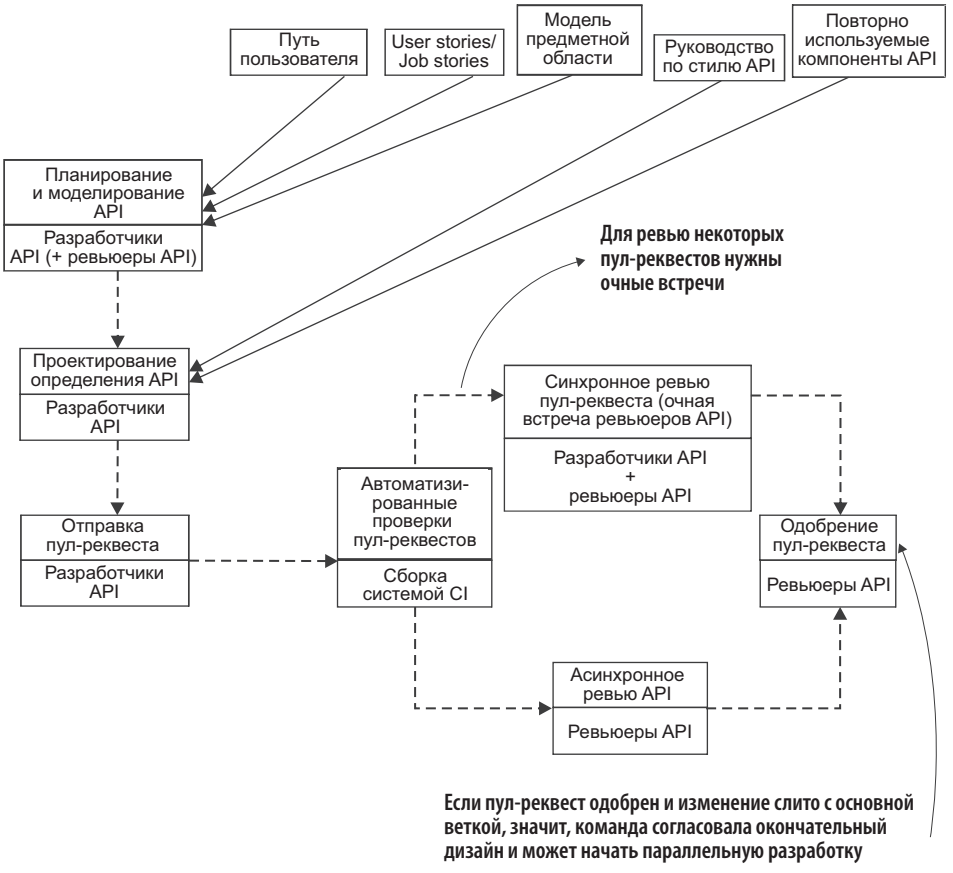


Рис. 5.2. Процесс дизайн-ревью API (пример). Ревьюеры могут посмотреть пул-реквест, оставить комментарии, одобрить его или организовать его очное, более обстоятельное обсуждение

Повышение единообразия и удобства использования API — основное преимущество ревью, однако есть и другие. Совместные ревью помогают синхронизировать и координировать действия внутренних команд, привлекая различных участников: тех, кого затрагивают изменения, и тех, кто в них заинтересован. Так, среди ревьюеров встречаются внутренние потребители API, тестировщики, разработчики и технические писатели. Команды получают шанс лучше понять суть изменений, поделиться своим мнением и выяснить, как правки повлияют на приложения или компоненты платформы, за которые они отвечают. Кроме того, во время ревью можно обучать сотрудников. Любой член команды, желающий развивать свои навыки разработки API, может присоединиться к обсуждению, чтобы узнать, как проектировать и анализировать интерфейсы.

https://liveinternet.club

> **СОВЕТ** В небольших организациях, где работает всего несколько команд разработчиков и сравнительно малое количество внешних API, скорее всего, нет отдельной группы, отвечающей за управление (API governance) и ревью. В таких условиях сложный процесс не потребуется. Иногда достаточно поработать над дизайном в паре с коллегой или обсудить решения в чате. Если API охватывает небольшое количество операций и относится к простой предметной области, то создание и ревью модели ресурсов API, описанное далее в главе, может оказаться излишним. Чтобы обеспечить подходящий уровень единообразия API, таким командам достаточно настроить простую систему автоматизированного линтинга.

Имейте в виду: хотя во время ревью разработчики API получают отзывы коллег, важно услышать и мнение более широкой аудитории, опубликовав новую версию для внешних потребителей API. В зависимости от уровня стабильности API (как обсуждалось в разделе 4.10.2), вы можете выпустить альфа- или бета-версию. Благодаря этому потребители смогут предварительно ознакомиться с дизайном, поэкспериментировать с ним и поделиться впечатлениями.

5.2. Проведение очного ревью API

Представьте, что вы — ревьюер API в команде управления. Одна из команд отправила пул-реквест на изменения в API. Ваши коллеги уже изучили его и решили, что нужно встретиться с разработчиками, чтобы обсудить некоторые проектные решения очно. Вас попросили организовать совещание для обсуждения запрашиваемых изменений в API. С чего бы вы начали?

Ниже представлен рекомендуемый формат встречи, который можно адаптировать с учетом ваших потребностей.

1. Назначьте встречу и пригласите участников.
2. В начале встречи озвучьте ее повестку.
3. Объясните ревьюерам бизнес-контекст изменений.
4. Проверьте модель ресурсов API.
5. Проверьте определение API и зафиксируйте отзывы ревьюеров.
6. Отметьте новые паттерны API и выявленные расхождения с руководством по стилю.
7. В конце встречи попросите участников заполнить короткий опрос.

Рассмотрим подробнее каждый из этих этапов, начиная с того, зачем вообще проводить ревью.

5.2.1. Назначение встречи и приглашение участников

Когда API передан ревьюерам, фасилитатор встречи должен пригласить всех заинтересованных участников — как тех, кто создает API, так и тех, кто будет его

https://liveinternet.club

использовать. Среди них могут быть представители разных ролей: инженеры, владельцы продукта, менеджеры по продукту, бизнес-аналитики, технические писатели и другие специалисты. Для удобства можно мысленно разделить участников ревью на три основные группы.

- *Разработчики API*. Как правило, это те, кто создает или обновляет файл определения API. Проектированием API могут заниматься несколько человек, и процесс подразумевает взаимодействие разных специалистов: архитекторов, менеджеров по продукту, его владельцев и т. д. В этой книге под *разработчиками API* понимается вся команда, участвующая в его создании или изменении.
- *Ревьюеры API*. Это те, кто рассматривает предложенные изменения в API. Среди них обязательно должны быть инженеры с большим опытом проектирования API. В крупных организациях ревьюерами могут быть члены команды управления. Также в число ревьюеров иногда входят представители API-платформы, ответственные за работу API-шлюза, ведущие специалисты по безопасности, технические писатели и сотрудники, которые отвечают за портал разработчика и опыт взаимодействия разработчиков с API.
- *Фасилитатор*. Человек, который организует и ведет очное ревью API. Он должен спланировать встречу, во время нее вести заметки и, при необходимости, осуществлять модерацию.

Важно заранее определить ключевых стейкхолдеров и привлечь их к ревью. Ниже приведены вопросы, которые помогут фасилитатору понять, кого следует пригласить.

- Кто будет интегрировать и потреблять API? Помимо внешних потребителей, могут быть и внутренние команды, которым полезно поучаствовать в проектировании используемого API.
- Кто будет создавать справочную документацию и руководства по работе с API? Возможно, технические писатели, которым предстоит документировать API, тоже хотят участвовать в ревью, чтобы лучше понять контекст и предложить правки для текстовых полей интерфейса.
- Есть ли среди участников ведущие специалисты по информационной безопасности (инженеры, тестировщики и т. д.), которые хотят проверить надежность и защищенность API?
- Задействованы ли команды, которые будут обеспечивать доступ к API через шлюз?
- Кто из инженеров и архитекторов решений заинтересован в работе API?
- Кто из владельцев и менеджеров продукта заинтересован в работе API?

Этот список вопросов не исчерпывающий, но он станет отличной отправной точкой. Дополнительно рекомендую делать очные ревью API открытыми для всех желающих, публикуя приглашения и напоминания во внутренних каналах коммуникации. Как уже упоминалось, очное ревью API позволяет обучать

https://liveinternet.club

сотрудников и обмениваться знаниями. Один из лучших способов подготовить будущих ревьюеров — открыть встречи для всех, кто в них заинтересован, хочет учиться и вносить свой вклад.

5.2.2. Представление повестки встречи

Если очное ревью API проводится официально, рекомендую фасилитатору начинать встречу с краткого изложения повестки. Это позволит участникам заранее понять структуру обсуждения и круг тем, которые планируется рассмотреть в отведенное время. Повестка помогает придерживаться основных вопросов и не отвлекаться на бесконечные дискуссии о проектировании. Помимо прочего, это хорошая возможность показать новым участникам, как обычно проводится ревью API.

Ниже вы увидите примерный план очного дизайн-ревью API, однако его следует адаптировать к условиям своей организации.

1. Убедиться, что на встрече присутствуют нужные стейкхолдеры.
2. Изложить тему ревью API.
3. Проверить модель ресурсов API.
4. Рассмотреть элементы REST API:
   - краткое и подробное описание;
   - требования к идемпотентности;
   - пути API;
   - успешные запросы и ответы;
   - сообщения об ошибках;
   - схему безопасности.
5. Отметить новые паттерны API и выявленные расхождения с руководством по стилю.
6. Провести завершающий опрос.

Благодаря единому формату повестки разные фасилитаторы смогут проводить очное ревью API более единообразно, а успешность обсуждения не будет зависеть от отдельных сотрудников.

5.2.3. Объяснение ревьюерам бизнес-контекста изменений

Убедившись, что собрались все необходимые участники (инженеры, специалисты по продукту и др.), перейдите к теме встречи. Чтобы ревьюеры лучше понимали, какое именно изменение будет рассматриваться, изложите бизнес-контекст, объясните, в чем заключается цель пользователя API, и, по возможности, покажите его путь (подробнее о пути пользователя рассказывается в разделе 5.3.1). Обычно, проводя встречу, я прошу разработчиков API кратко, за 3–5 минут, представить

https://liveinternet.club

бизнес-контекст изменения, после чего даю участникам возможность задать вопросы, чтобы все поняли его суть.

Другой способ дать участникам общее представление об API — показать профиль API, объясняющий его назначение. Пример представлен в табл. 5.1.

Таблица 5.1. Пример профиля API

Объект	Описание
Название API	API каталога продуктов
Ценностное предложение API	API открывает доступ к широкому ассортименту товаров компании Acme и позволяет выбирать средства для ухода за домашними питомцами
Потребитель API	Конечный пользователь — клиент, просматривающий каталог Acme Pet Supplies с целью покупки. Разработчики клиентской части будут использовать API для создания трех каналов взаимодействия с конечным пользователем: мобильного приложения Acme, интернет-магазина Acme и магазина партнеров
Job stories	1 Когда я изучаю доступные товары Acme, я хочу просматривать каталог по категориям, чтобы находить новые и существующие интересные мне товары для животных 2 Когда я изучаю доступные товары Acme, я хочу найти в каталоге определенный товар для животных, чтобы посмотреть информацию о нем и принять решение о покупке Примечание. Возможность поиска по каталогу будет реализована на этапе 2
Уровень доступа	Общедоступный API, который могут использовать как собственные цифровые платформы Acme, так и зарегистрированные партнеры
План использования	Мобильное приложение Acme: 10 запросов в минуту Интернет-магазин Acme: 10 запросов в минуту Магазины партнеров: 8 запросов в минуту
Модель безопасности	Запросы к API должны проходить авторизацию с использованием протокола OAuth2. В начальной версии с минимальной функциональностью будет разрешено только чтение данных, но по мере добавления новых функций появятся и другие права доступа

Альтернативный способ сообщить рецензентам бизнес-контекст — использовать *шаблон ценностного предложения* API. Он демонстрирует преимущества API, показывая, как функциональные возможности интерфейса решают боли потребителей, приносят им желаемую выгоду и помогают выполнять задачи.

https://liveinternet.club

Инструмент особенно полезен на ранних этапах проектирования API: он служит основой для обсуждений с инженерами, архитекторами, менеджерами по продукту, спонсорами и другими стейкхолдерами, помогая согласовывать, что именно должен делать API.

Job story и цифровые возможности

Job story — расширенная версия пользовательской истории (user story) в концепции *Jobs to Be Done (JTBD)* — «работа, которая должна быть выполнена». Job story подразумевает фокус на инициировании ситуации, мотивации пользователя и ожидаемом результате, то есть цели, которую он пытается достичь. Например, в профиле API каталога продуктов из табл. 5.1 job story может звучать так: «Когда мне нужно средство для ухода за питомцем, я хочу увидеть широкий выбор товаров с хорошими отзывами, чтобы выбрать среди них тот, который мне понравится».

JTBD — это методика, с помощью которой определяется задача, или «работа» (job), для выполнения которой клиент «нанимает» продукт или услугу. Благодаря такому подходу менеджеры по продукту выявляют истинные потребности и мотивы, побуждающие клиентов приобретать товар или услугу. Потребитель будет «нанимать» API для решения конкретной задачи, и ее четкое описание позволит менеджеру спроектировать интерфейс наилучшим образом. Опираясь на job story, команда сможет определить *цифровые возможности системы* — основные требования, которым должен удовлетворять API, чтобы выполнить эту задачу. Подробнее о том, как определять требования, область развития и границы API, анализируя job stories и цифровые возможности системы, вы найдете в книге Джеймса Хиггинботама «Principles of Web API Design» (Addison-Wesley, 2021).

Шаблон ценностного предложения API включает в себя два основных раздела. Слева находится профиль потребителя (или клиента), где описываются задачи, которые он хочет выполнить с помощью API, проблемы, с которыми столкнулся, и потенциальная выгода. Справа располагается карта ценности — в ней перечислены доступные ресурсы: внутренние API и источники данных, приложения и процессы, которые можно объединить для создания API, способного решить проблему потребителя или принести ему выгоду. Эти два раздела соединяются в интерфейсе — специальном поле, где формируется ценность API для потребителя (рис. 5.3).

Заполняя карту ценностного предложения на этапе проектирования API, вы можете следовать нумерации на рис. 5.3. Рисунок 5.4 демонстрирует частично заполненную карту ценностного предложения для API каталога продуктов. В следующем разделе мы заполним блок об операциях и ресурсах API.

Шаблон ценностного предложения API описан в книге Андреа Зулиан и Амансио Бузы «API Product Management» (Leanpub, 2017), где авторы называют его *value proposition interface canvas* (шаблон интерфейса ценностного предложения). Это адаптация шаблона Александра Остервальдера (Alexander Osterwalder,

https://liveinternet.club

www.strategyzer.com/canvas/value-proposition-canvas). Заполнение шаблона также происходит на первом этапе метода Марьюкки Ниинойи APIOps Cycles.

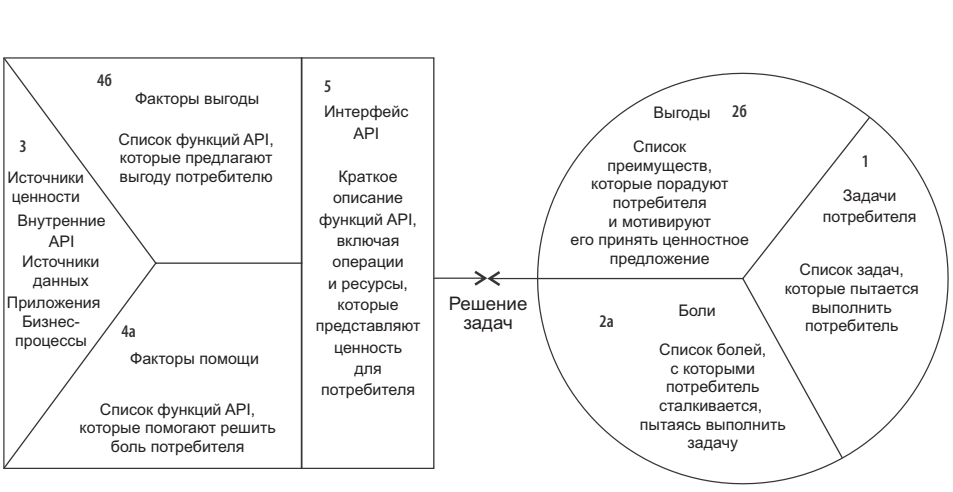


Рис. 5.3. Основные разделы карты ценностного предложения API. Источник: адаптация шаблона ценностного предложения из книги Андреа Зулиан и Амансио Бузы (Andrea Zulian, Amancio Bouza) «API Product Management» (Leanpub, 2017)



Рис. 5.4. Шаблон ценностного предложения для API каталога продуктов (Products Catalog API). API получает необходимые данные из внутренних API микросервисов Products, Catalog и Reviews

https://liveinternet.club

Основы метода APIOps Cycles

APIOps Cycles (www.apiopscycles.com/the-method) — метод совместного создания API-решений и стратегий развития API, основанный на принципах бережливого производства. Он разработан Марьюккой Ниинойя из компании Osaango. Проектирование API следует начинать с тщательного анализа потребностей компании и бизнес-модели, которую он должен поддерживать. Метод помогает организации определить:

- какие API ей необходимы;
- как проектировать новые API;
- как совершенствовать существующие API.

Метод APIOps Cycles состоит из восьми этапов.

- *Этап 1. Описание бизнес-контекста с помощью шаблона.* Сначала заполните шаблон ценностного предложения API, шаблон бизнес-модели API и карту пути клиента, чтобы стейкхолдеры (архитекторы, владельцы решений, менеджеры по продукту и бизнес-аналитики) могли определить, какие API необходимо разработать и почему. Участники смогут обсудить и согласовать, какие API обеспечат максимальную ценность компании, кто будет целевым потребителем, какие преимущества должен предлагать API, какова стоимость его разработки и как он повлияет на доход компании. Эти вопросы связаны с ценностным предложением API и его влиянием на бизнес-модель.
- *Этап 2. Учет опыта разработчиков.* Стейкхолдеры определяют, как помочь целевым потребителям обнаружить и использовать API, а также как организация будет развивать и поддерживать сообщество разработчиков.
- *Этап 3. Согласование минимально жизнеспособной архитектуры API.* Определите минимальную, но работоспособную архитектуру, которая обеспечит соблюдение нефункциональных требований к каждому API. Это поможет оптимизировать график работы API.
- *Этап 4. Разработка API.* Спроектируйте модели данных и схемы взаимодействий, создайте прототип API и реализуйте API согласно руководству по стилю, принятому в организации. При этом нужно учитывать требования к данным: степень их актуальности, международные и отраслевые стандарты и общепринятые идентификаторы.
- *Этап 5. Аудит API.* Проверьте прототип API, его дизайн или продакшен-версию с помощью чек-листов, чтобы убедиться, что API соответствует стандартам организации.
- *Этап 6. Публикация API.* Стейкхолдеры определяют, какие материалы должны быть опубликованы (включая документацию по продукту, техническую документацию и спецификации) и как помочь потребителям обнаружить и использовать API.
- *Этап 7. Отслеживание, измерение, анализ.* Определите и отслеживайте бизнес-показатели и технические метрики, чтобы оценивать производительность и успешность внедрения API.
- *Этап 8. Обучение и улучшение.* На основе качественных и количественных данных о работе API совершенствуйте как сам API, так и процесс его разработки.

Подробнее об этапах APIOps Cycles можно узнать из книги Яркко Мойланена, Марьюкки Ниниойя и Марко Сеппянена (Jarkko Moilanen, Marjukka Niinioja, Marko Seppänen) «API Economy 101» (Books on Demand, 2019). Она послужит хорошим дополнением к материалам на официальном сайте.

https://liveinternet.club

5.2.4. Проверка модели ресурсов API

Рекомендую по возможности проводить проверку операций и моделей данных, которые планируется реализовать в API. Если разработчики и стейкхолдеры согласуют однозначную модель API, это поможет быстрее решать вопросы, которые обычно возникают при создании определения API. Причем важно не только обсудить, но и задокументировать эту модель в процессе проектирования. Так, в книге «Principles of Web API Design» (Addison-Wesley Professional, 2021) Джеймс Хиггинботам предлагает процесс проектирования API, основанный на методике ADDR (Align-Define-Design-Refine, «согласовать, определить, спроектировать, уточнить»). Это подход design-first, включающий семь шагов.

1. Определите способности системы, которые должны помогать клиентам достигать желаемых результатов.
2. Опишите действия, которые пользователь должен выполнить, чтобы достичь этих результатов.
3. Определите границы API, сгруппировав способности системы.
4. Создайте модель профиля API, указав, какие ресурсы и операции должен предоставлять API.
5. Создайте высокоуровневый дизайн API с учетом выбранного архитектурного стиля (например, REST, GraphQL или gRPC).
6. Доработайте дизайн API в соответствии с обратной связью, чтобы сделать его удобнее для пользователей.
7. Подготовьте документацию: справочные материалы и руководства по началу работы с API.

На четвертом шаге, создавая модель профиля API (концептуальную модель данных и операций, лежащих в основе API), привлекайте специалистов разных ролей и из разных команд. Совместное дизайн-ревью API и согласование результатов помогут сформировать единый взгляд на модель данных и универсальную терминологию для ее описания. Эту модель и связанные с ней функции можно наглядно представить в виде ER-диаграмм (Entity-Relationship, сущности и связи), таблиц и других визуальных инструментов.

В качестве примера рассмотрим простой API каталога продуктов. Его модель данных включает три основных ресурса: Products (товары), Reviews (отзывы) и Category (категория). Это и есть те сущности, которые раскрывает REST API. У товара может быть любое количество отзывов (в том числе ни одного), при этом каждый отзыв относится исключительно к одному товару. Товар может принадлежать к одной или нескольким категориям, а в категории может быть любое количество товаров (в том числе ни одного). Отношения между ресурсами показаны на рис. 5.5 с использованием нотации «воронья лапка».

https://liveinternet.club

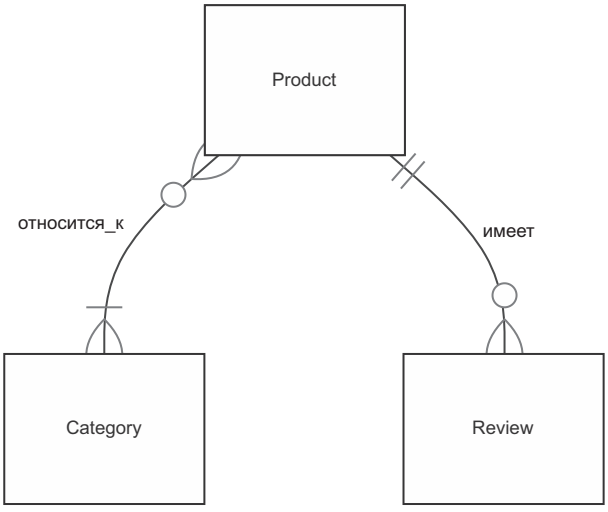


Рис. 5.5. Отношения между веб-ресурсами, поддерживающими операции API каталога продуктов

> **СОВЕТ** Нотация «воронья лапка» — это способ представления ER-диаграммы, при котором множественность связи обозначается символом в виде трех ответвлений. Подробнее о нотации можно узнать на странице https://vertabelo.com/blog/crow-s-foot-notation/.

Допустим, этот API также поддерживает операции `listCategories`, `listProducts`, `viewProduct`, `listProductReviews` и `viewReview`, как показано в табл. 5.2. Когда модель API определена, команда может использовать ее как ориентир, чтобы оценивать, правильно ли функции и данные преобразованы в RESTful API на основе OpenAPI.

Помимо таблицы, перечисленные операции и ресурсы можно вписать в блок API на карте ценностного предложения, как показано на рис. 5.6, и тем самым завершить шаблон.

Таблица 5.2. Операции API для API каталога продуктов

Название операции	Описание операции	Участник	Веб-ресурс	Запрос	Ответ
`listCategories`	Перечисляет все категории	Клиент	Category	Фильтр по полю, сортировка по полю, направление сортировки, размер страницы, указатель страницы	Category

https://liveinternet.club

Название операции	Описание операции	Участник	Веб-ресурс	Запрос	Ответ
`listProducts`	Перечисляет все товары	Клиент	Products	Фильтр по полю, сортировка по полю, направление сортировки, размер страницы, указатель страницы	Products
`viewProduct`	Показывает информацию о товаре	Клиент	Products	ID товара	Products
`listProductReviews`	Перечисляет все отзывы о товаре	Клиент	Reviews	Фильтр по полю, сортировка по полю, направление сортировки, размер страницы, указатель страницы	Reviews
`viewReview`	Показывает отзыв на товар	Клиент	Reviews	ID товара, ID отзыва	Reviews

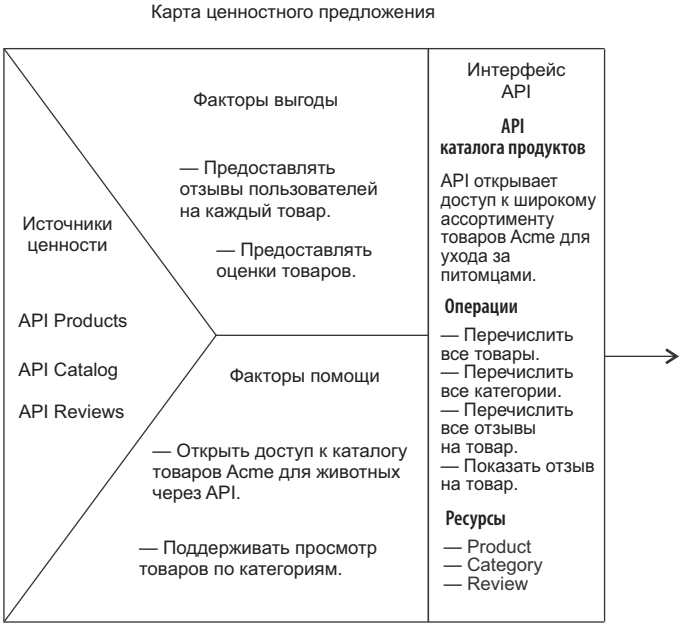


Рис. 5.6. Левая часть в шаблоне ценностного предложения API каталога продуктов теперь заполнена и содержит операции и ресурсы

https://liveinternet.club

Итак, вы определили высокоуровневые требования и желаемую модель API. Но как новый API будет интегрирован в микросервисную архитектуру Acme? API-шлюз компании отвечает за проверку подлинности и перенаправляет все запросы, прошедшие аутентификацию, в сервис API каталога продуктов. Этот сервис реализует интерфейс продукта API и оркестрирует запросы к бэкенд-микросервисам товаров, категорий и отзывов. Они, в свою очередь, предоставляют данные, с помощью которых API каталога продуктов может формировать и выдавать ответы. Взаимодействие компонентов показано на рис. 5.7.

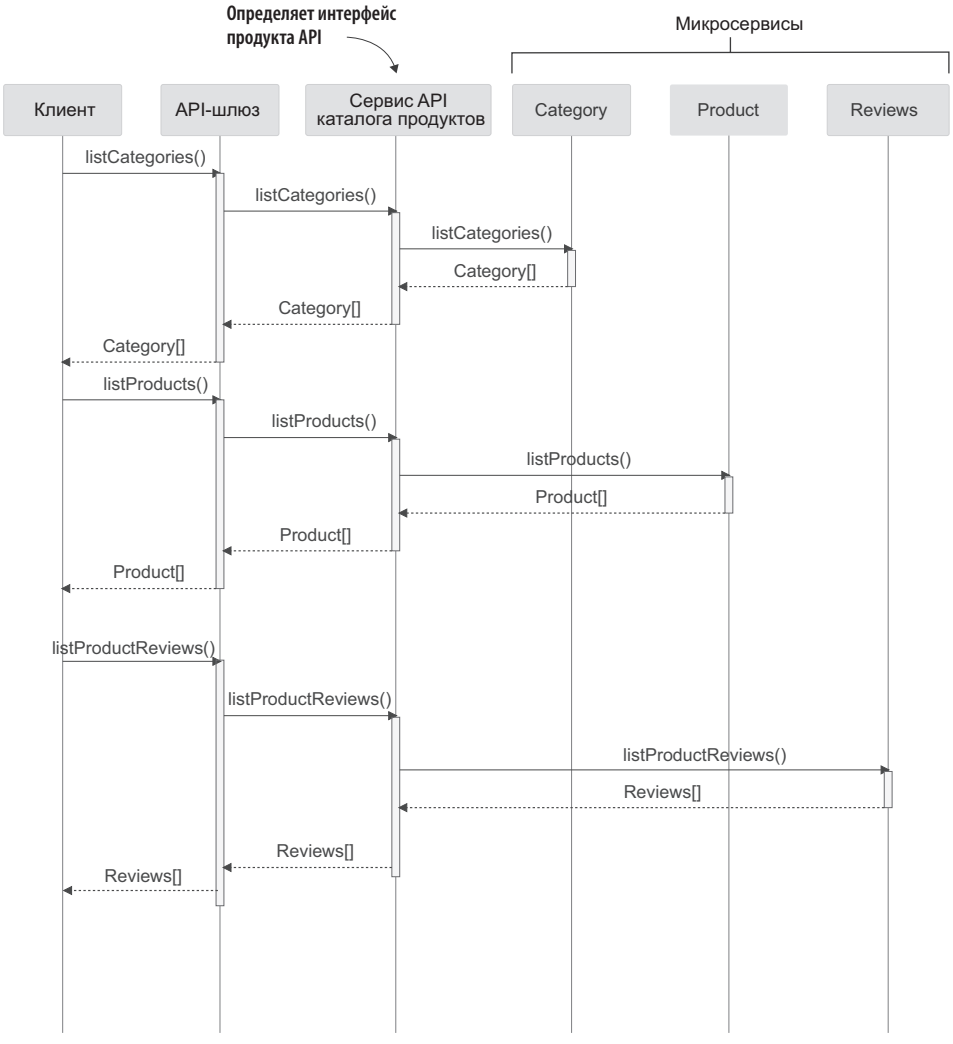


Рис. 5.7. Сервис API каталога продуктов оркестрирует запросы к бэкенд-микросервисам

https://liveinternet.club

Узнав требования к операциям и модели ресурсов API, ревьюеры получают четкое представление о предметной области и готовы оценивать файлы определения API. Хотя приведенный пример довольно упрощен, на практике полезно иметь подобный контекст: он помогает ориентироваться в предметной области и сопоставлять ее элементы с операциями REST API.

5.2.5. Проверка определения API

Пришло время тщательно проанализировать пул-реквест на изменение API. Если команда успешно провела автоматизированный линтинг и поиск критических изменений, ревьюерам не нужно перепроверять полученные результаты, ведь одна из целей автоматизации — сократить время реализации на этапе ревью API. Так что ревьюерам лучше сосредоточиться на руководстве по стилю API и общих проблемах дизайна, которые можно устранить только вручную. Следует изучить результаты автоматизированных проверок, уделяя внимание предупреждениям и информационным сообщениям (проблемам уровней INFO и WARN). Кроме того, если пул-реквест рассматривается повторно, важно не упустить комментарии, оставленные во время первой проверки.

Я советую перед ревью составить чек-лист проверок. Если большинство рекомендаций из руководства по стилю API уже соблюдаются благодаря автоматизированным проверкам, нужно будет лишь рассмотреть вручную оставшиеся пункты. Однако если в организации пока не внедрили полный набор программных проверок для определений API, ревьюерам придется тщательно проконтролировать соответствие API всем требованиям руководства. Тратить время ревьюеров на такую задачу — не лучшая идея (именно поэтому важны линтинг и автоматизация!), и команда, отвечающая за управление API, должна автоматизировать как можно больше подобных проверок. Вот пример чек-листа.

- *Посмотрите отчеты автоматизированных проверок.* Убедитесь, что линтинг и проверки на наличие критических изменений завершены успешно, а в отчетах не осталось предупреждений, требующих дополнительного внимания.
- *Каким будет путь пользователя и как предложенные изменения в API помогут пользователю выполнить задачу?* Просмотрите справочную документацию API. Понятно ли в ней описана цель создания API и задача, которую он помогает решить пользователю?
- *Соответствует ли модель веб-ресурсов принятой модели предметной области?* Проверьте, охватывает ли API операции и ресурсы, указанные в требованиях. Все ли операции на месте? Не выходят ли они за рамки задач API? Отметьте любые несоответствия и непонятные названия атрибутов и ресурсов и укажите на проблемы, оставив комментарии к пул-реквесту. Выполняя эту проверку, вы получаете отличную возможность обсудить с коллегами все расхождения в терминологии, относящейся к ресурсам и атрибутам. Идея единого языка, предложенная в предметно-ориентированном проектировании, применима и к разработке API.

https://liveinternet.club

- *Является ли API расширяемым?* Подумайте, как будет развиваться API: расширяя существующие операции или добавляя новые.
- *Понятны ли описание API, краткие пояснения и примеры использования?* Прочитайте общие сведения об API в объекте info, описание каждой операции для эндпоинтов и текст, сопровождающий атрибуты сообщений. Проверьте, правильно ли изложена информация и будет ли она понятна разработчикам, интегрирующим API. Убедитесь также, что все примеры значений в определении API максимально полезны и реалистичны.
- *Все ли важные сообщения об ошибках возвращаются с соответствующими кодами ошибок?* Проверьте, во всех ли операциях используются корректные HTTP-коды ошибок и все ли важные сценарии ответа учтены. Кроме того, убедитесь, что пользовательские поля ошибок в теле ответа содержат значимую информацию.
- *Соблюдаются ли в операциях API требования к идемпотентности?* Если эндпоинт идемпотентен, один и тот же запрос можно отправлять многократно, при этом состояние на сервере изменится только один раз. Убедитесь, что операции API идемпотентны сами по себе либо используют ключ идемпотентности, если необходимо предотвратить дублирование.

  ПРИМЕЧАНИЕ Согласно спецификации HTTP, методы `GET`, `PUT` и `DELETE` по умолчанию идемпотентны, а `POST` и `PATCH` — нет. Если операция, требующая идемпотентности (например, создание платежа), реализована через POST или PATCH, обязательно используйте *ключ идемпотентности*, чтобы замечать и правильно обрабатывать повторяющиеся запросы.

- *Рассмотрите нефункциональные требования к API.* Рассмотрите безопасность, разбивку на страницы и свойства дизайна, способные повлиять на производительность приложения. Помимо механизмов аутентификации и авторизации, нужно учитывать и другие требования к безопасности, касающиеся структуры и валидации запросов и ответов. Проверка безопасности с помощью линтинга рассматривается в главе 9.

Во время ревью API я обычно прошу участников оставлять замечания в комментариях к пул-реквесту, и самые важные вопросы мы обсуждаем вместе. Такой подход помогает фасилитатору отслеживать состояние каждой выявленной проблемы.

5.2.6. Выявление новых паттернов API и расхождений с руководством по стилю

Очное ревью API — отличная возможность выявить новые паттерны API, которые стоит добавить в руководство по стилю, принятое в организации. Кроме того, на встречах иногда выявляются двусмысленности и недочеты в текущем

https://liveinternet.club

руководстве, из-за которых разработчики могут неправильно обрабатывать некоторые элементы API. Фасилитатор должен фиксировать подобные находки, чтобы затем обсуждать их с соответствующими подразделениями, например с командой управления или внутренними профессиональными сообществами. В результате руководство по стилю и правила линтинга могут быть изменены. Во время ревью важно внимательно слушать участников, чтобы выявлять пробелы в руководстве по стилю и устранять их.

5.2.7. Завершающий опрос

Чтобы дальнейшие обсуждения проходили еще эффективнее, в завершение встречи фасилитатор может попросить ревьюеров ответить на несколько вопросов. Подробнее об этом я расскажу в разделе 5.5.

Итак, мы изучили основную структуру встреч для очного дизайн-ревью API. Теперь рассмотрим некоторые трудности, с которыми обычно сталкиваются команды во время ревью.

5.3. Трудности ревью API

В процессе ревью существует ряд трудностей. Во-первых, нужно постоянно прилагать усилия, чтобы менее опытные разработчики и ревьюеры API смотрели на продукт извне, с точки зрения будущего пользователя. Во-вторых, ревью требует больше ресурсов, если API оказался некачественным. Кроме того, команды сталкиваются с проблемами, если ревью проводится на слишком позднем этапе, а иногда бывает трудно определить подходящий уровень проверки.

5.3.1. Принятие подхода, ориентированного на пользователя

Менее опытным разработчикам бывает нелегко рассматривать API с точки зрения пользователя. Во время проектирования они стремятся упростить реализацию бэкенда и ради этого жертвуют удобством потребителей API и качеством документации. В ходе ревью значительная часть усилий уходит на то, чтобы помочь командам взглянуть на API глазами тех, кто будет с ним взаимодействовать. Для этого нужно начать с карты пути конечного пользователя и двигаться назад к технической реализации (рис. 5.8).

Один из способов этого добиться — попросить специалистов отправлять на проверку не только файл определения Open API, но и раннюю версию API и справочной документации. Написание документации помогает разработчикам взглянуть на API с точки зрения конечного потребителя.

СОВЕТ Более подробную информацию о картах сервиса можно найти на странице Nielsen Norman Group: https://mng.bz/aEeB.

https://liveinternet.club

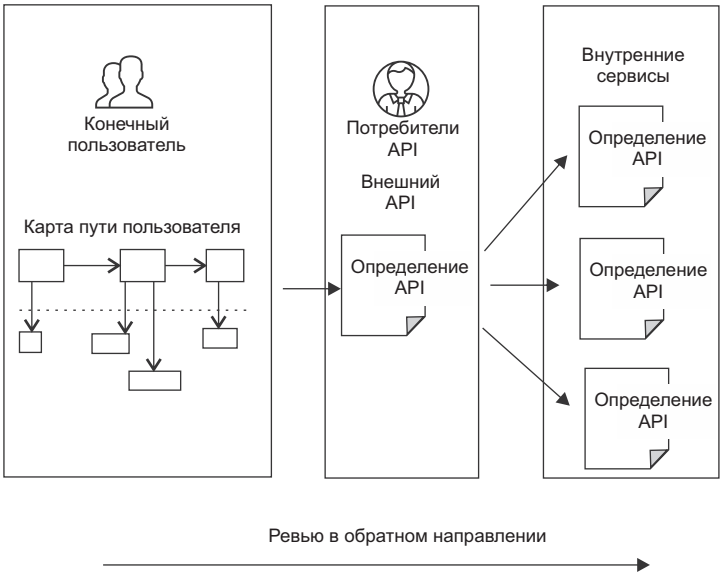


Рис. 5.8. В ходе ревью внешнего API по возможности начните с карты пути пользователя, двигаясь в обратном направлении. Предполагается, что карта уже готова. Проверив внешний API, переходите к анализу внутренних сервисов, от которых он зависит

Еще один полезный инструмент для развития мышления, ориентированного на пользователя, — это карта пути. *Карта пути пользователя* (также «карта пути клиента») помогает наглядно представить, как человек взаимодействует с продуктом или сервисом для достижения своей цели или выполнения задачи. Путь рисуется с точки зрения действующего лица (персоны). В нем описан сценарий использования, поставленная цель, основные и дополнительные этапы, а также конкретные действия пользователя на каждом из них. Карту пути можно дополнить картой сервиса — в ней описаны компоненты (например, API поставщика), с которыми пользователь взаимодействует. Карта сервиса помогает ревьюерам переключиться с внутренней перспективы, где решения зависят от процессов и систем в организации, на внешнюю, ориентированную на действия и мысли пользователей.

С помощью карты сервиса разработчики и ревьюеры могут сосредоточиться на действиях и возможностях пользователей на каждом этапе пути, что позволяет создавать расширяемые API, способные при необходимости в будущем адаптироваться к новым требованиям.

Однако, если вы хотите обращаться к такой карте во время проверки API, разработчики должны подготовить ее заранее. Именно здесь пригодится метод APIOps Cycles. Его первый этап — описание бизнес-контекста с помощью шаблона — как раз и помогает команде продумать путь пользователя.

https://liveinternet.club

5.3.2. Некачественное моделирование перед рецензированием

Чтобы правильно проектировать REST API, необходимо прежде всего понимать, что представляют собой ресурсы API, какие между ними взаимосвязи и какие операции можно с ними выполнять. Если разработчикам не хватило опыта для качественного моделирования API, то на этапе ревью, скорее всего, будет выявлено нарушение стандартов REST. Это может привести к долгим, утомительным обсуждениям. Но подобную ситуацию можно предотвратить, если помочь командам с моделированием API на более ранних этапах, что приводит нас к следующему пункту.

5.3.3. Слишком позднее ревью

Хотя очное ревью и автоматические проверки помогают обнаружить проблемы с единообразием и удобством использования API, иногда они проводятся слишком поздно — когда дизайн API уже предложен. Ревьюеры принесут больше пользы, если подключатся к работе на раннем этапе проектирования, предложив разработчикам инструменты и рекомендации для создания качественного API. Например, они смогут обеспечить доступ к руководству по стилю API и линтерам, а также повысить качество моделирования API и проектирования сервисов. Ни ревью, ни автоматизированные проверки не заменят своевременного взаимодействия с командами разработчиков.

5.3.4. Определение уровня ревью

Иногда команды задаются вопросом: стоит ли применять единые правила управления ко всем категориям API — внутренним, партнерским и общедоступным? Давайте подумаем. Автоматизированные проверки уже встроены в пайплайн, и их масштабирование на дополнительные файлы определения API обходится дешевле, поэтому целесообразно применять их ко всем возможным API. В свою очередь, для ревью нужно тщательно координировать и планировать работу отдельных специалистов и целых команд, и оно требует более значительных расходов из-за затрат времени ревьюеров и разработчиков. Есть простое и надежное правило: проводить очное ревью только для внешних API (общедоступных и партнерских) и тех, которые могут стать таковыми в будущем. Внутренние API, не предназначенные для общего доступа, можно проверять менее строго (или вовсе исключить из ручных проверок, если того захотят разработчики).

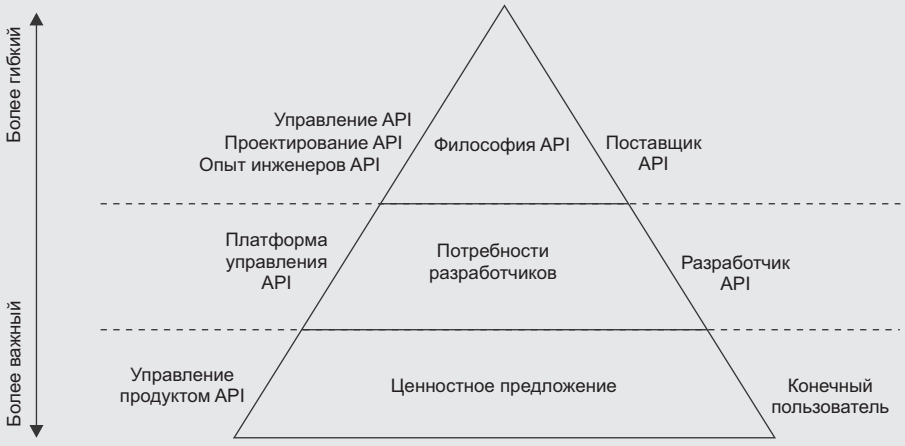
Иерархия принципов проектирования API

В книге «API Product Management» Андреа Зулиан и Амансио Буза представили иерархию принципов проектирования API. Авторы утверждают, что в основе успешного API лежит четкое ценностное предложение для клиента, то есть конечного пользователя, который применяет функции API для выполнения своих задач. При этом главная задача менеджера по продукту — помочь определить и подтвердить это предложение. Следующий уровень предполагает создание удобной среды для разработчиков, которые настраивают интеграцию с API. Им требуется исчерпываю-

https://liveinternet.club

щая документация, возможность простого доступа (в том числе получение ключей API и другие необходимые механизмы аутентификации), тестовая среда (песочница), SDK и прочие вспомогательные ресурсы. Чтобы их предоставить, нужны платформа управления API и портал разработчика.

На третьем уровне иерархии находится философия API. Она предполагает, что при проектировании API нужно учитывать ограничения, связанные с ценностным предложением и потребностями разработчиков, а также применять принципы управления API, руководства по стилю и накопленный опыт инженеров. Иерархия продемонстрирована на следующем рисунке.



Иерархия принципов проектирования API (источник: книга Андреа Зулиан и Амансио Бузы «API Product Management»)

Андреа и Амансио подчеркивают, что дизайн API должен в первую очередь сохранить ценностное предложение для клиента и удовлетворить потребности разработчиков. Если не удается найти баланс между тремя принципами и нужен компромисс, авторы рекомендуют сохранить базовые уровни и корректировать философию API. Предложенную иерархию полезно учитывать при проведении ревью.

5.4. Оценка эффективности ревью

Предположим, вы внедрили автоматизированные проверки пул-реквестов на внесение изменений в API и на случай необходимости ввели очное ревью. Как измерить эффективность этих процессов?

5.4.1. Автоматизированные проверки

Чтобы оценить эффективность автоматизированных проверок, можно использовать рейтинг качества API: в нем учитывается количество ошибок линтинга, критических изменений и случаев несоответствия (о соответствии

https://liveinternet.club

API поговорим в главах 6 и 7). Если команда небольшая, можно сохранять эти данные как артефакты сборки на CI-сервере. А вот крупным командам, вероятно, понадобится отдельная панель мониторинга API, которая будет собирать и визуализировать метрики качества и единообразия.

5.4.2. Очное ревью API

Чтобы повысить продуктивность очного ревью API, важно получать обратную связь от разработчиков API касательно того, насколько оно полезно. Рекомендую выбрать один из двух подходов: собирать отзывы сразу после встречи или опрашивать коллег с определенной периодичностью. В первом случае фасилитатор, завершая обсуждение, предлагает разработчикам API анонимно пройти короткий опрос — к примеру, по специальной ссылке. В нем достаточно всего двух пунктов: пятибалльной шкалы для оценки пользы (например, «Насколько встреча была полезной для повышения качества API?») и открытого вопроса для сбора комментариев («Что можно улучшить?»).

Во втором случае вы запрашиваете у разработчиков API более развернутое мнение, но делаете это реже — скажем, один-два раза в год. Такие опросы хорошо проводить в формате шкалы Ликерта, где участникам нужно оценить общее качество API, удобство использования, документацию API, единообразие и работу с критическими изменениями. Как и при первом подходе, советую добавить открытый вопрос для сбора комментариев. Пример опроса:

1. Дизайн-ревью API помогает повысить общее качество наших API.
   - Совершенно не согласен
   - Не согласен
   - Затрудняюсь ответить
   - Согласен
   - Совершенно согласен
2. Дизайн-ревью API помогает повысить удобство использования наших API.
   - Совершенно не согласен
   - Не согласен
   - Затрудняюсь ответить
   - Согласен
   - Совершенно согласен
3. Дизайн-ревью API помогает повысить качество документации API.
   - Совершенно не согласен
   - Не согласен
   - Затрудняюсь ответить

https://liveinternet.club

   - Согласен
   - Совершенно согласен

4. Дизайн-ревью API помогает повысить соответствие API руководству по стилю, принятому в организации.
   - Совершенно не согласен
   - Не согласен
   - Затрудняюсь ответить
   - Согласен
   - Совершенно согласен

5. Дизайн-ревью API помогает командам избежать критических изменений.
   - Совершенно не согласен
   - Не согласен
   - Затрудняюсь ответить
   - Согласен
   - Совершенно согласен

6. Поделитесь своими мыслями о том, что можно улучшить.

Сравнивая результаты опросов за разные периоды, можно понять, совершенствуется ли процесс ревью.

Помимо опроса разработчиков, ревьюеры API или команда управления (если она есть в организации) могут проводить встречи и ретроспективы, чтобы оценить и улучшить процесс ревью. Кроме того, в зависимости от размера команды управления и количества команд, которым она помогает, одним из ключевых показателей эффективности (KPI) ревьюеров может быть число незакрытых пул-реквестов. Статистика запросов обычно доступна в отчетах облачных сервисов контроля версий, например GitHub и Bitbucket. Команда ревьюеров должна быстро обрабатывать пул-реквесты и сокращать среднее время, в течение которого пул-реквест ожидает проверки.

5.5. Документирование процесса дизайн-ревью API

Чтобы эффективность ревью не падала при большом масштабе, важно задокументировать процесс. Это снизит риск недопонимания и путаницы, обеспечит единообразие проверок, упростит обучение новых сотрудников, а также поможет анализировать и совершенствовать сам процесс ревью.

В крупных организациях, разрабатывающих множество API, как правило, есть централизованная команда управления — она следит за качеством и единообразием дизайна внешних API и формирует процесс ревью, который интерфейсы должны проходить перед публикацией.

https://liveinternet.club

Команды разработчиков ПО существенно различаются по характеру и масштабу проектов API, внутренней культуре, принятым подходам и другим параметрам, поэтому не существует универсальных правил ревью, подходящих всем без исключения. Каждая команда должна адаптировать руководство под собственные задачи и потребности. Тем не менее в следующих разделах я предложу несколько рекомендаций, которые помогут задокументировать этот процесс.

5.5.1. Обеспечьте легкий доступ к документу

Разместите документ о проведении ревью API там, где его легко найти, например, создайте внутреннюю вики-страницу или файл с общим доступом. Опрос в моей организации показал: большинство стейкхолдеров предпочитают, чтобы описание процесса (и руководство по стилю API) находилось в одном репозитории с определениями API. Где бы он ни хранился, нужно обеспечить быстрый доступ к этому документу и возможность создавать на него ссылки, потому что команды будут регулярно к нему обращаться.

5.5.2. Сформулируйте принципы и цели

Опишите ценности и принципы, которыми нужно руководствоваться при проведении дизайн-ревью API. К примеру, организация может отдать приоритет разработке интерфейсов, которые будут простыми и понятными для потребителей, вместо тех, что точно отражают ее внутренние технологические системы и структуру. В соответствии с принципами Agile, можно определить, что важнее быстро изменять API в ответ на потребности компаний, чем строго придерживаться заранее утвержденного дизайна или плана. Возможно, вы считаете, что при подготовке внешних API необходимо следовать подходу design-first, при котором разработка интерфейса начинается только после согласования дизайна, благодаря чему потребитель получает более качественный и удобный API, а поставщик повышает эффективность разработки и снижает расходы. Еще один принцип гласит, что ревьюеры и, главное, потребители API должны без труда понимать назначение и функциональность интерфейса, прочитав справочную документацию и руководства. Для этого документы должны быть достаточно простыми, но при этом достаточно подробными. Приведенные принципы — лишь примеры. Подумайте, какие ценности действительно вам важны, и внесите в руководство именно их. Исходя из наиболее распространенных принципов, дизайн-ревью API должно стремиться к достижению следующих измеримых целей.

- *Качество*. Повышайте качество API и обеспечьте его соответствие стандартам.
- *Скорость*. Сократите время, необходимое для проверки и утверждения дизайна.
- *Надежность*. Объясните стейкхолдерам, чего ожидать, составьте график ревью и не отклоняйтесь от него, чтобы руководители проекта, менеджеры по продукту и отдел продаж могли планировать работу, выполнять обязатель-

https://liveinternet.club

ства перед клиентами и соблюдать сроки доставки. Для этого важно заранее выделить достаточное количество ревьюеров, поскольку они — основной ресурс процесса.

- *Гибкость*. Процесс ревью следует адаптировать для различных сценариев. Необходимо эффективно обрабатывать как небольшие изменения API (например, расширение эндпоинта в существующем ресурсе), так и крупные (скажем, при разработке совершенно нового API со множеством эндпоинтов и новых ресурсов). Кроме того, важно иметь возможность изменять приоритеты ревью в зависимости от приоритетов доставки.
- *Стоимость*. Процесс не должен сопровождаться большими затратами. К ревью API привлекаются ключевые инженеры, архитекторы, менеджеры по продукту, эксперты по безопасности и другие специалисты. Длительное совещание с одновременным участием всех стейкхолдеров обходится дорого, поэтому результат ревью должен оправдывать расходы. Значит, важно минимизировать потери, причиной которых служат ошибки в дизайне API, задержки, недопонимание, подготовка избыточной информации и ненужные встречи.

5.5.3. Опишите метрики процесса

В разделе 5.4 я привел некоторые KPI, которые помогут измерить эффективность ревью. Их обязательно стоит внести в документ, описывающий процесс, чтобы стейкхолдеры понимали, как оценивается результативность, и могли ее отслеживать.

5.5.4. Распределите роли и обязанности

Если ревью масштабное, следует однозначно определить, кто и за какую его часть отвечает. Как уже упоминалось в разделе 5.2.1, основные участники обсуждения — разработчики API, ревьюеры и фасилитаторы.

> **СОВЕТ** Назначать роли и обязанности можно при помощи матрицы распределения ответственности (RACI). Это таблица, в которой описывают поставленные задачи и для каждой из них указывают *исполнителя*, *ответственного*, *консультанта* и *информируемое лицо* (Responsible, Accountable, Consulted, Informed). Подробную информацию о таблице RACI вы найдете на сайте www.mindtools.com/agn584l/the-raci-matrix.

Так, фасилитатор может отвечать за административные задачи и исполнять их: планировать встречи, актуализировать пул-реквесты, тикеты и статусы проверки. Специалист с этой ролью также назначает ревьюеров, контролирует сроки и помогает стейкхолдерам найти информацию о процессе ревью. Если организация небольшая, штатный фасилитатор не обязателен.

От ревьюеров API требуется обратная связь по результатам — желательно, в комментариях к пул-реквесту, где ее можно посмотреть в будущем. В крупных организациях могут работать несколько штатных ревьюеров, тогда как

https://liveinternet.club

в небольших эту роль иногда выполняют добровольцы команды или временно назначенные эксперты. В любом случае ревьюеры должны выделять время на проверки и иметь права на одобрение изменений в репозитории API.

В свою очередь, разработчики API могут предлагать изменения в API, обсуждать рекомендации ревьюеров и вносить согласованные правки.

5.5.5. Проводите очное ревью по запросу

Допустим, у вас настроен пайплайн CI/CD, автоматически проверяющий пул-реквесты на внесение изменений в API, а разработчики прикладывают к запросам черновики справочной документации, которая позже будет опубликована на портале. В таком случае очные обсуждения достаточно проводить, только если об этом попросят сами ревьюеры или разработчики. Как правило, встречи требуются, если не удается решить проблему в ветке комментариев к пул-реквесту. В идеале ревьюеры оценивают предлагаемые изменения в API, анализируя файл определения и сопроводительную документацию. Не забывайте: когда потребители API пытаются понять, как работает интерфейс, они в основном ориентируются на документацию, поэтому и ревьюеры должны полагаться на нее, а не ожидать устных разъяснений при встрече. Как уже упоминалось, длительные совещания с большим количеством участников — принимая в расчет ценность времени каждого из них — требуют значительных затрат, поэтому встречи следует проводить только при необходимости.

5.5.6. Категоризируйте API по уровню ревью

Прежде чем проводить ревью, бывает полезно узнать, какой уровень контроля требуется для разных категорий API в организации. Так вы поймете, нужно ли привлекать к обсуждению специалистов команды управления. Уровни ревью для каждой категории удобно представить в виде таблицы (см. табл. 5.3).

Таблица 5.3. Определение необходимого уровня ревью для разных категорий API

Категория API	API для внешнего использования	Автоматизированные проверки в пайплайне CI/CD	Ревью пул-реквеста командой разработчиков	Ревью пул-реквеста командой управления
Внутренний API	Нет	Да	Да	Нет
Внутренний API (может быть опубликован)	Да	Да	Да	Не обязательно, но рекомендуется
Партнерский API	Да	Да	Да	Да
Внешний API	Да	Да	Да	Да

https://liveinternet.club

5.5.7. Определите явный этап утверждения

В рамках подхода design-first, документируя процесс очных ревью, для всех стейкхолдеров нужно четко указать действие, которое подтверждает, что изменения в дизайне приняты и API переходит со стадии проектирования на стадию внедрения. Например, зафиксировать, что ревью считается завершенным, если одобрен пул-реквест и правки слиты с основной веткой, либо изменился статус тикета. Это важно для параллельной разработки, ведь пока разработчики не будут уверены, что дизайн утвержден, они не начнут его внедрять.

5.5.8. Формализуйте внесение изменений после утверждения

Утверждение дизайна не означает, что его больше нельзя менять. Если в процессе внедрения выясняется, что нужны правки, разработчик API может создать новый пул-реквест и проинформировать об изменениях ревьюеров и других стейкхолдеров (команды фронтенда и т. д.). Такой запрос проверяется и принимается в обычном порядке, а если изменения небольшие, то ревью не занимает много времени. В отличие от водопадной методологии, в Agile приветствуется адаптация. Один из принципов Agile-манифеста гласит: «Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества» (подробнее на странице https://agilemanifesto.org/principles.html).

5.5.9. Определите, как обозначать исключения

Иногда, чтобы внести изменение, приходится отклониться от принятого в организации руководства по стилю API — например, если нужно расширить устаревшие API, созданные до его написания. Чтобы обеспечить такую адаптивность, укажите в описании процесса, как разработчикам API обозначать и аргументировать свое решение отклониться от руководства по стилю. Один из способов — отметить затрагиваемые компоненты API с помощью специального расширения OpenAPI. При этом важно задокументировать причину, по которой понадобилось исключение. Фасилитаторы очного ревью также должны решить, нужно ли обновить правила линтинга, чтобы игнорировать такое изменение.

5.5.10. Определите способ приоритизации ревью

Небольшие команды редко сталкиваются с такой задачей, но в организациях с большим количеством API фасилитаторам иногда приходится определять приоритетность пул-реквестов. В документе, описывающем процесс ревью API, должны быть зафиксированы однозначные критерии приоритизации — например, на основе целевой даты релиза или доставки. Они помогут обеспечить надежность процесса. В некоторых системах для совместной работы можно назначать приоритеты с помощью меток в пул-реквестах. Если же запрос связан с тикетом на устранение проблемы, достаточно использовать метку приоритета самого тикета.

https://liveinternet.club

5.5.11. Укажите ожидаемый срок ревью

Чтобы обеспечить надежность процесса, важно указать, сколько времени команда управления будет рассматривать пул-реквест от разработчиков API. Это поможет менеджерам по продукту и руководителям проекта лучше планировать работу и формировать реалистичные ожидания у клиентов и других стейкхолдеров. Сроки ревью можно классифицировать по характеру изменений. Пример представлен в табл. 5.4.

Таблица 5.4. Сроки ревью разных изменений API

Тип изменения	Продолжительность ревью (рабочие дни)
Расширение существующих ресурсов для имеющихся эндпоинтов	Не более одного дня
Добавление нового эндпоинта для существующего ресурса	Не более двух дней
Добавление новых ресурсов и новых эндпоинтов	Не более пяти дней

5.5.12. Обеспечьте прозрачность процесса ревью

В некоторых крупных организациях пул-реквест привязывают к тикету на устранение проблемы, благодаря чему команда управления может отслеживать статус тикета на специальной доске. Другие команды добавляют к запросам метки приоритета. Эти меры повышают прозрачность процесса для разработчиков API и других стейкхолдеров. Однако при этом фасилитаторы ревью должны следить за тем, чтобы статусы тикета и пул-реквеста всегда были актуальны. Рассмотрим пример возможных статусов на доске.

- *Бэклог*. Пул-реквест создан, ожидается назначение ревьюеров и определение приоритета.
- *Назначено* или *запланировано*. Пул-реквест назначен ревьюерам, или его ревью добавлено в расписание.
- *На проверке*. Ревьюеры API рассматривают пул-реквест.
- *Запрошены изменения*. Ревью завершено, но ревьюеры оставили рекомендации, и разработчики должны прокомментировать их или исправить API.
- *Утверждено*. Изменение в дизайне принято, контракт API согласован. Стейкхолдеры могут использовать его как основу для работы.
- *Отклонено*. Пул-реквест отклонен с описанием причины.

5.5.13. Опишите, как стать ревьюером API

Если над API организации работает много команд разработчиков, обычно также существует центральная команда управления со штатными членами, среди

https://liveinternet.club

которых ревьюеры с большим опытом проектирования и проверки API. Тем не менее полезно привлекать к процессу добровольцев из других команд — так обратная связь будет разнообразнее. Некоторые специалисты, например эксперты по безопасности или технические писатели, могут внести особый вклад в ревью дизайна и документации API благодаря своим профессиональным знаниям. Чем больше таких участников, тем выше качество, скорость и надежность ревью. Поэтому важно информировать сотрудников о возможности стать ревьюерами API и четко описать в документации процесса, как присоединиться к команде проверяющих.

5.6. Примеры ревью API

В предыдущих разделах я описал упрощенный порядок ревью API. Теперь стоит рассмотреть, как этот процесс устроен в организациях, разрабатывающих множество интерфейсов. Обратимся к опыту Microsoft Azure и Google.

5.6.1. Управление API в Microsoft Azure

Подразделение разработки в Microsoft Azure следует концепции ответственного управления API. Такой подход предполагает сотрудничество с внутренними командами разработчиков с самых ранних этапов, чтобы помочь им внедрить лучшие практики проектирования интерфейсов. Главная цель — сформировать культуру, которая поможет создавать простые в использовании, единообразные и масштабируемые API, соответствующие стандартам отрасли. За процесс отвечает специальная команда ответственного управления API (API Stewardship team) под руководством опытных специалистов, среди которых инженеры сервисов Azure, архитекторы SDK, сотрудники офиса технического директора, специалисты по работе с клиентами и др. При этом к обсуждению может присоединиться любой сотрудник организации, заинтересованный в API. Это открытая группа, объединяющая профессионалов из разных областей и увлеченных добровольцев.

В подходе команды управления можно выделить три основных направления работы. Во-первых, она взаимодействует с разработчиками с ранних этапов проектирования API, обучает их этому процессу и помогает сформировать собственные руководства по стилю, документацию API и политику работы с критическими изменениями. Кроме того, опытные специалисты делятся вспомогательными материалами по общим вопросам проектирования API и сервисов.

Во-вторых, команда управления предоставляет инфраструктуру и инструменты для автоматизированной проверки дизайна API (в том числе линтинга и выявления критических изменений) сразу при отправке пул-реквеста, а также для генерации артефактов API — клиентских библиотек, мобильных SDK, мок-серверов, документации REST API и многого другого. В результате две

https://liveinternet.club

трети правил проверяются автоматически в пайплайне разработки, а одна треть контролируется вручную во время дизайн-ревью API.

В-третьих, для помощи командам организуется дизайн-ревью и выпуск предварительных версий API. Во время очного дизайн-ревью в Microsoft Azure команда управления API первым делом просит разработчиков объяснить предметную область и цель API. Затем участники обсуждения рассматривают примеры запросов и ответов в файлах определения OpenAPI через редактор Swagger. Они ищут типичные ошибки проектирования, например неправильную реализацию идемпотентности или разбивки коллекций на страницы. После этого обсуждают другие важные проектные решения, связанные с потоком выполнения кода, будущим развитием API, полиморфизмом, пакетными операциями и т. д. К дизайн-ревью обязательно привлекаются менеджеры по продукту, поскольку некоторые решения могут влиять на продуктовую стратегию. Команда управления API также помогает организовать выпуск предварительных версий интерфейса — экспериментальных, ранних сборок API, с которыми потребитель может настроить интеграцию. Это позволяет собирать обратную связь от партнеров и первых пользователей. Команда управления помогает разработчикам определить четкие цели для этапа предварительного тестирования, чтобы точно знать, на какие результаты взаимодействия с API обратить внимание.

> **СОВЕТ** Подробную информацию о модели ответственного управления в Microsoft Azure вы найдете в выступлении Майка Кистлера и Марка Вайцеля (Mike Kistler, Mark Weitzel) «Building APIs at Scale: Moving from API governance to API stewardship» (www.youtube.com/watch?v=sboH7zyfEnA) и презентации Адриана Хэла (Adrian Hal) «Lessons Learned on the Azure API Stewardship Journey» (https://mng.bz/gv0E).

Чтобы реализовать систему ответственного управления API, в Microsoft Azure выделили штатных специалистов, в том числе менеджера по продукту и архитектора, а также подготовили необходимые материалы и инструменты. Главная цель — помочь командам создать качественный API еще на раннем этапе проектирования, а не контролировать и исправлять дизайн API после выпуска, когда вносить существенные изменения бывает уже поздно.

5.6.2. Обработка предложений по улучшению API в Google

Специалисты Google проводят дизайн-ревью API, чтобы убедиться, что разрабатываемые интерфейсы просты, интуитивно понятны и соответствуют другим API компании. Руководство по стилю API в Google — API Improvement Proposals, AIP — лаконично задает общие принципы реализации. Оно доступно на сайте https://aip.dev. Кроме того, Google предоставляет линтер (https://linter.aip.dev) для автоматической проверки API на соответствие AIP. Однако не все требования AIP можно выразить в виде правил линтинга, поэтому разработчикам API все равно нужно внимательно изучить руководство.

https://liveinternet.club

Прежде чем запустить API, с которым пользователи Google смогут взаимодействовать посредством кода, необходимо провести ревью и утвердить дизайн — это обязательное требование для выпуска бета-версии (готовой для публичного тестирования и стабильной) и общедоступной сборки (открытой для всех клиентов и готовой к использованию на продакшен). При этом дизайн-ревью не требуется для API, предназначенных исключительно для сотрудников или работы внутренних инструментов компании.

В команду управления API входят несколько штатных ревьюеров, однако в дизайн-ревью также участвуют большое количество инженеров с опытом проектирования API и желанием помочь. Во время ревью специалисты Google анализируют API с точки зрения неопытного пользователя: изучают доступные действия, ресурсы и сопроводительную документацию. Они также следят за тем, чтобы API был расширяемым и сохранял единообразие с другими интерфейсами Google. Обычно дизайн-ревью API в компании проходит асинхронно, через внутренний инструмент Google для проверки пул-реквестов (которые также называются списками изменений). Но если возникают трудности, которые невозможно решить с помощью обсуждения в ветке запроса, разработчики API могут обратиться к ревьюерам и назначить получасовую встречу.

> **СОВЕТ** Чтобы узнать, как в Google внедряли дизайн-ревью API для решения проблем с качеством и единообразием, прочитайте статью «API Design Reviews at Scale» (CHI'16 Extended Abstracts, 2016; https://research.google/pubs/pub45294), подготовленную исследователями Маквин, Мэйли и Доутри (Macvean, Maly и Daughtry) из Google.

Описание процесса дизайна-ревью API в Google дает понять, сколько времени занимает проверка изменений. Анализ инкрементных правок в существующем API потребует несколько дней, а обсуждение новых, но небольших API — около недели. Ревью более крупных интерфейсов может проводиться дольше.

5.6.3. Дизайн-ревью API в Slack

Проектирование API в Slack начинается с того, что команда разработчиков определяет его предполагаемые сценарии использования и цели изменения API. Затем специалисты создают файл определения API, который содержит предложенный API, и размещают его во внутреннем канале #api-decisions для проверки. Канал объединяет межфункциональную группу ревьюеров и стейкхолдеров из разных подразделений: инженеров, менеджеров по продукту, специалистов по взаимодействию с разработчиками и экспертов по безопасности. В некоторых случаях для утверждения API достаточно асинхронной проверки в канале. В других — требуется более обстоятельное обсуждение, и тогда разработчики назначают очное ревью в рамках рабочих часов, выделенных на рассмотрение API.

После внутреннего ревью, перед тем как специалисты приступают к разработке, файл определения API должен пройти еще одну проверку. К ней привлекается

https://liveinternet.club

группа избранных партнеров, которые предоставляют дополнительную обратную связь. После ее обработки команда реализует API и открывает некоторым партнерам доступ к нему для предварительного просмотра и бета-тестирования. Замечания, полученные на этом этапе, используются для дальнейшего улучшения API. Благодаря такому подходу специалисты Slack выявляют и устраняют проблемы еще до выпуска общедоступной версии. Подробную информацию о процессе ревью API в Slack вы найдете в публикации «How We Design Our APIs at Slack» (https://mng.bz/eoGv).

5.6.4. Дизайн-ревью API в Kubernetes

Kubernetes (https://kubernetes.io/) — это платформа с открытым исходным кодом, предназначенная для управления контейнеризованными приложениями. Она служит для запуска больших облачно-ориентированных приложений, предоставляя механизмы для их развертывания и масштабирования в зависимости от доступных ресурсов (процессора, памяти и т. д.). В управляющем слое Kubernetes есть API-сервер, который предоставляет REST API для управления объектами Kubernetes, благодаря чему клиенты могут настраивать приложения и контейнеры. Этот API отличается большим масштабом и включает в себя сотни эндпоинтов. Чтобы обеспечить его расширяемость, разработчики разделили все ресурсы на *группы API*, каждая из которых имеет свою версию. Ресурсы основной группы *core* (также называется *устаревшей*) размещаются в пути с префиксом `/api/v1/` (например, `GET /api/v1/namespaces/test/pods`). Другие группы используют префикс `/apis/$GROUP_NAME/$VERSION` (например, `/apis/batch/v1/`).

В описании процесса дизайн-ревью API в Kubernetes (https://mng.bz/ppX8) рассказывается, как сообщество проекта обеспечивает единообразие и удобство использования API. Четко определено, какие изменения API требуют обязательного ревью (к примеру, изменения API из основной группы), а какие проверяются по желанию. У проекта есть руководство по стилю API, называемое *API Conventions* (Соглашения API). Прежде чем отправить пул-реквест на изменения в API, разработчик должен изучить это руководство и убедиться, что не нарушил требования. Кроме того, в запросе нужно указать цель изменения и ссылку на утвержденное предложение по улучшению (Kubernetes Enhancement Proposal, KEP) — документ, аналогичный описанию новой функции или требований.

Ревьюеры и фасилитаторы API регулярно просматривают поступающие запросы. Если пул-реквест соответствует требованиям проекта, его добавляют на панель ревью, где можно отслеживать его статус и нагрузку на ревьюеров. Фасилитаторы ревью API (в Kubernetes их называют модераторами) расставляют приоритеты запросов в бэклоге. Приоритет определяется с учетом нескольких факторов: этапа релиза, на который рассчитано изменение, мнения руководителей проекта, объема и сложности вносимых правок, а также уровня стабильности новой версии (порядок приоритетов: общедоступная версия > бета > альфа). Благодаря этому можно в первую очередь проверять наиболее

https://liveinternet.club

важные и срочные изменения. Для проверки каждого пул-реквеста модератор назначает ревьюера исходя из его загруженности и компетенций в предметной области. Кроме того, модераторы берут на себя административную работу, чтобы ревьюеры могли уделить больше времени анализу API, и при необходимости помогают организовать очное обсуждение.

Попав на панель ревью, запрос проходит несколько этапов: Бэклог > Назначено > В работе > Запрошены изменения > Ревью завершено. Если пул-реквест прошел ревью и был утвержден, на него впоследствии ссылаются в пул-реквесте на внесение изменений по реализации, чтобы ревьюеры могли проверить, насколько код соответствует утвержденному дизайну. В редких случаях запрос на ревью может быть отклонен — например, если изменение должно быть реализовано вне проекта Kubernetes.

Описание процесса дизайн-ревью API в Kubernetes помогает разработчикам понять, сколько времени требуется для проверки. В нем указано, что расстановка приоритетов и составление расписания занимают не более недели, а завершение первого ревью — до трех недель.

Чтобы стать ревьюером API в Kubernetes, необходимо в совершенстве освоить руководство по стилю, а также хорошо понимать структуру проекта и архитектуру системы. Новички участвуют в предварительных проверках под руководством наставников, которые обучают их и содействуют их включению в число официальных ревьюеров.

Итоги

- Ревью API в рамках подхода APIOps подразумевает проверку пул-реквеста на изменения в API после того, как он прошел автоматизированные проверки. Главная задача ревьюера — выявить проблемы, которые не обнаружили программные инструменты.
- Ревью API проводится очно или асинхронно, а ревьюеры могут оставлять комментарии в ветке пул-реквеста.
- Во время ревью рекомендуется ориентироваться на потребителя: начать с версии определения API, которую он будет использовать, и затем двигаться в обратном направлении — к внутренним интерфейсам. Перед началом ревью убедитесь, что вы понимаете контекст изменений.
- В крупных организациях, как правило, в ревью участвуют разработчики, ревьюеры и фасилитатор. При этом к обсуждению могут присоединиться другие заинтересованные специалисты.
- Чтобы оценить эффективность очного ревью API, можно провести опрос среди его участников. Проверить качество самого API помогут отчеты автоматизированных проверок.

https://liveinternet.club

6 Соответствие API: генерация кода и определений API

В этой главе

- ✓ Решение задачи соответствия API (API conformance) в коде серверной части, клиентской части и документации
- ✓ Генерация кода на стороне сервера для обеспечения соответствия
- ✓ Генерация определений OpenAPI для обеспечения соответствия
- ✓ Генерация SDK для обеспечения соответствия

Представьте, что вы создаете общедоступный REST API. Вы спроектировали определение OpenAPI и провели линтинг и проверку критических изменений, как описано в главах 3 и 4. Вы построили и развернули свой API на API-шлюзе и привели его в рабочее состояние. Также вы сгенерировали для API справочную документацию и опубликовали ее на портале разработчика API. Пока все нормально.

Но потребители вашего API жалуются на то, что некоторые подробности опубликованной документации не соответствуют реальному поведению API. Например, в документации API указано, что один из эндпоинтов всегда возвращает некоторое поле в ответе 200 OK. Но когда потребители реально получают ответ от API, поле в нем отсутствует. Между справочной документацией и развернутым API существует расхождение. Что нужно сделать, чтобы

https://liveinternet.club

API на сервере и клиентская реализация SDK соответствовали определению OpenAPI?

Справочная документация API представляет собой контент на портале разработчика, который описывает предоставляемые операции API в формате, понятном для человека. Как правило, документация API генерируется на основе определения OpenAPI. Для этого можно воспользоваться программными инструментами или отправить определение OpenAPI на портал разработчика, который сгенерирует документацию API за вас. Из документации потребители API узнают, что можно делать с вашим API. А значит, если какие-то части документации не соответствуют реальному поведению API, это породит путаницу и ухудшит впечатления разработчиков. Они начнут открывать тикеты на исправление дефектов, повышая ваши затраты на поддержку. А при масштабировании проблемы с документацией могут повредить массовому распространению вашего API.

Существуют два основных способа обеспечить соответствие сервера API, клиентских SDK и документации API определению OpenAPI. В первом варианте (генеративное соответствие) либо код серверной заглушки API генерируется по определению OpenAPI, либо определение OpenAPI генерируется на основании кода сервера API. В обоих случаях клиентские SDK и документация API генерируются исходя из созданного определения OpenAPI. Под *заглушкой* (stub) здесь понимаются генерируемые программные объекты, представляющие сообщения запроса и ответа в вызове API, а также логику контроллера и маршрутизации. Второй подход — тестирование на соответствие — основан на запуске тестов (включая проверки времени выполнения в API-шлюзе), которые проверяют, что ответы сервера API соответствуют определению OpenAPI.

В этой главе мы сосредоточимся на первом способе, и вы узнаете, как использовать методы генеративного соответствия в своих рабочих потоках APIOps. Обычно это происходит на стадии сборки рабочего потока APIOps (рис. 6.1).

В этой главе рассматривается, как сгенерировать код серверных заглушек, определений OpenAPI и клиентов SDK, чтобы они гарантированно были согласованы. Кроме того, рассматриваются некоторые обстоятельства, которые необходимо учитывать для поддержания синхронизации генерируемых артефактов. В приложении Г приведена информация о том, как настроить инструменты и код, необходимый для этой главы, включая сведения об установке, запуске и остановке API-шлюза Kong. Прежде чем читать дальше, убедитесь, что вы подготовили все необходимые инструменты. Чтобы работать с упражнениями этой главы, перейдите в каталог `chapter6` репозитория с кодом книги по адресу https://github.com/apiopsbook/apiops. Материал этой главы в большей степени актуален для разработчиков, но обзор задачи соответствия API также может оказаться полезным для владельцев продуктов API.

https://liveinternet.club

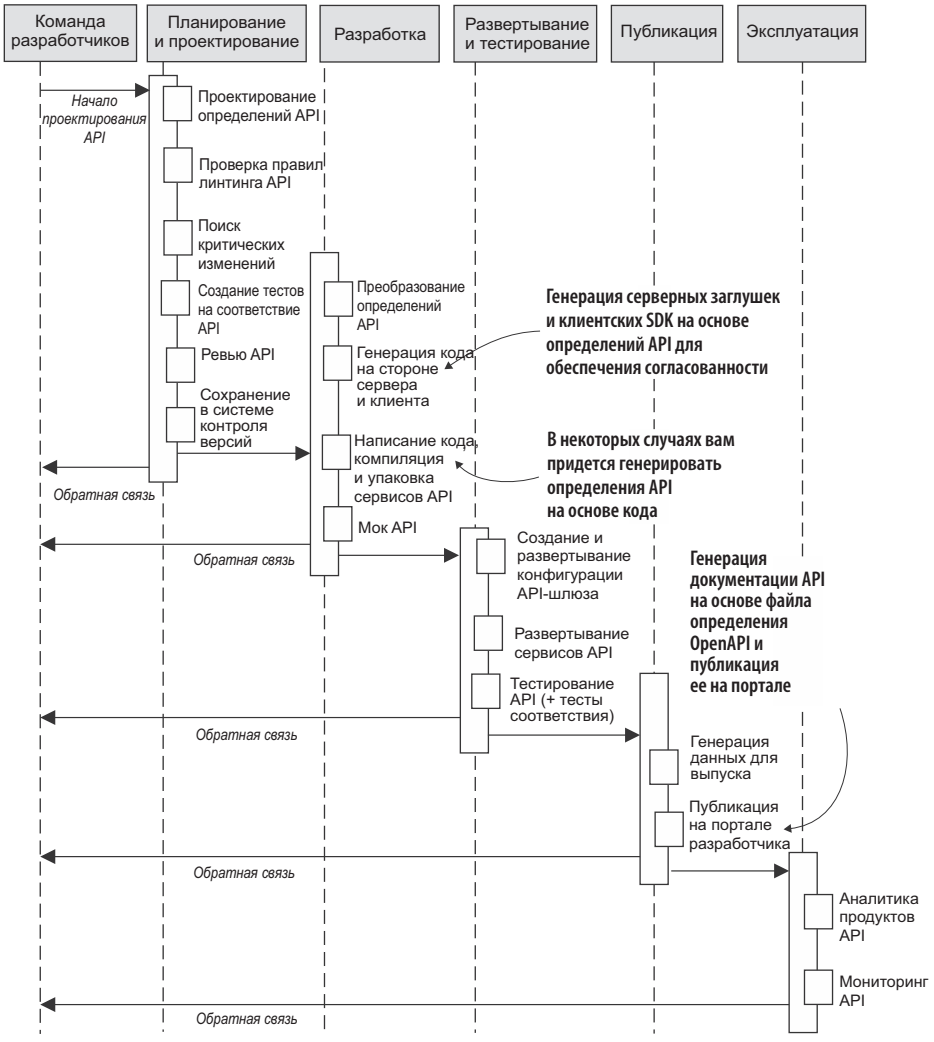


Рис. 6.1. На стадии сборки можно сгенерировать код серверной заглушки и клиентских SDK по определению API. Определения API преобразуются в документацию API и публикуются на портале разработчика

6.1. Задача соответствия API

Возьмем следующий сценарий. Пользователи заметили несоответствия между опубликованной документацией API для эндпоинта — допустим, `GET /v1/catalog/products/{id}` — и поведением эндпоинта на сервере. В ответе отсутствует конкретное поле `keywords`, заявленное в документации API как обязательное

https://liveinternet.club

(рис. 6.2). На иллюстрации видно, что сервис API получает запрос от API-шлюза и передает его различным микросервисам для обработки.

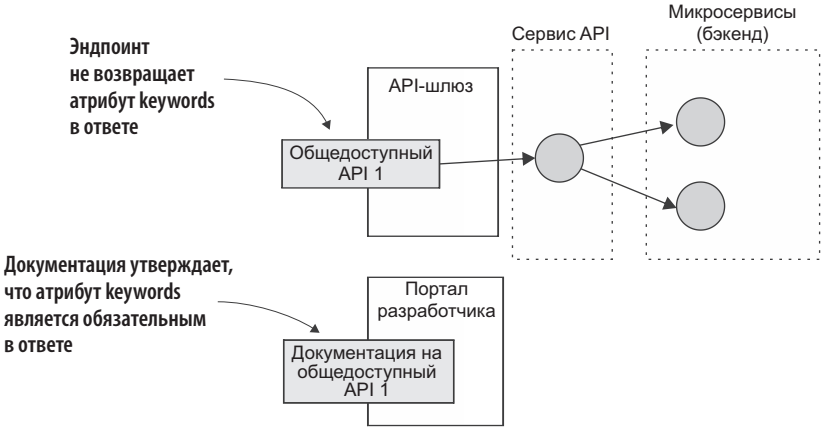


Рис. 6.2. В ответе общедоступного API отсутствует поле, помеченное как обязательное в документации API. Возникает несоответствие между документацией и работающим API

Как это произошло? Есть различные варианты. Возможно, разработчик сервиса API написал логику контроллера вручную, взяв определение OpenAPI с этапа проектирования за образец, поэтому мог упустить тот факт, что атрибут `keywords` должен присутствовать в ответе. Определение OpenAPI с этапа проектирования является источником истины относительно того, что должен делать API, но разработчик создавал код контроллера сервиса API вручную, и это привело к человеческим ошибкам в преобразовании определения OpenAPI в код. А может быть, в ходе процесса разработки изменились требования, и атрибут `keywords` перестал быть обязательным. Разработчик обновил код, но забыл обновить файл определения OpenAPI. В таком случае команда разработки не рассматривала определение OpenAPI как источник истины для обновления кода.

Как бы то ни было, в итоге разработчик развернул на API-шлюзе сервис API, но опубликовал документацию, основанную на исходном определении API. Между реализацией API и опубликованным определением OpenAPI возникло расхождение (рис. 6.3).

Должен существовать механизм, который обеспечивал бы синхронизацию кода контроллера и определения OpenAPI. Конечно, разработчик может постоянно помнить о необходимости обновлять исходное определение API при появлении нового требования. Проблема в том, что это зависит от человеческой памяти, а обновления необходимо вносить в двух местах — в коде и определении API. Если разработчик забудет обновить определение API (на

https://liveinternet.club

основе которого публикуется документация API), будет невозможно узнать, что существует проблема расхождения, пока кто-то вручную не проверит документацию API на согласованность или потребитель API не откроет тикет на исправление ошибки.

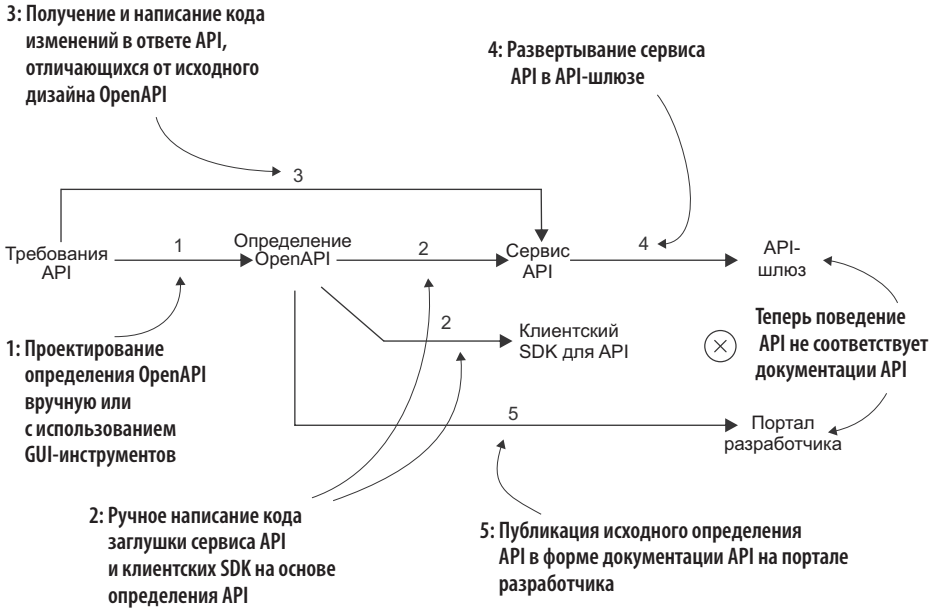


Рис. 6.3. Изменения в API на этапе разработки могут привести к расхождению между поведением развернутого API и исходным определением

Другой вариант — обновление только в одном месте (определении OpenAPI) и генерация кода контроллера сервиса API, клиентского SDK и документации API на основе этого определения. Иначе говоря, определение OpenAPI с этапа проектирования не только рассматривается как источник истины, но и минимизирует риск человеческих ошибок за счет автоматической генерации кода заглушки сервиса API, клиентского SDK и документации на основании этого определения. Такой подход требует инструментов, генерирующих код заглушки сервиса API для заданного языка программирования и фреймворка приложений на основе определений OpenAPI. Каждый раз, когда разработчик получает требование на изменение запроса или ответа API, он сначала обновляет определение OpenAPI, а затем заново генерирует на его основе заглушку контроллера сервиса API. Эта схема представлена на рис. 6.4.

А теперь посмотрим, как это сделать в проекте Java с использованием библиотеки OpenAPI Generator в качестве инструмента для генерации кода серверной заглушки.

https://liveinternet.club

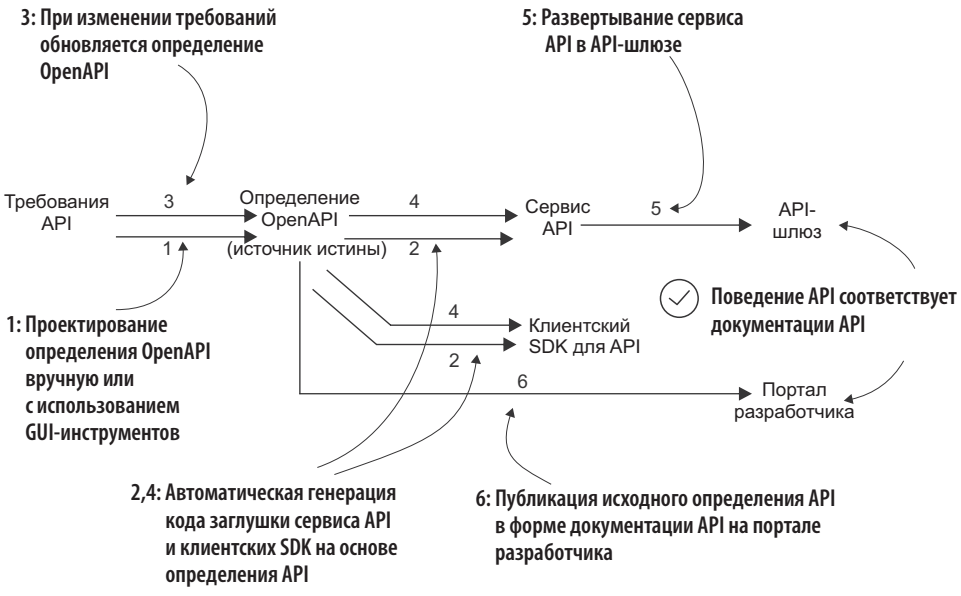


Рис. 6.4. Обновление определения OpenAPI при каждом изменении требований API с последующей генерацией заглушки контроллера сервиса API на основе определения OpenAPI. Такой подход гарантирует, что реализация будет соответствовать опубликованной документации API

6.2. Генерация кода на основе OpenAPI

Чтобы обеспечить соответствие поведения API определению OpenAPI, можно рассматривать последнее как авторитетную спецификацию API и генерировать на его основе объекты контроллера сервиса API, запроса и ответа при помощи инструментов генерации кода. В этом разделе я покажу, как это делается, на примере приложения Java Spring Boot. Не беспокойтесь, если вы не знакомы с Java и Spring Boot, — я создал примеры, которые можно просто запустить. А если вы владеете Spring Boot, разделы с описанием структуры проекта можно пропустить. Хотя я достаточно подробно описываю файлы, создаваемые библиотекой генерации кода, в этом разделе я не пытаюсь обучить вас Java или Spring Boot либо показать рекомендуемый вариант структурирования приложений Spring Boot. Я хочу просто представить пример того, как генерировать код контроллера, модели и маршрутизации для сервиса, используя определение OpenAPI в качестве источника истины.

6.2.1. Spring Boot с OpenAPI

Представьте, что тимлид предлагает вам при создании API использовать генерацию кода. Технические требования указывают, что сервис API должен быть

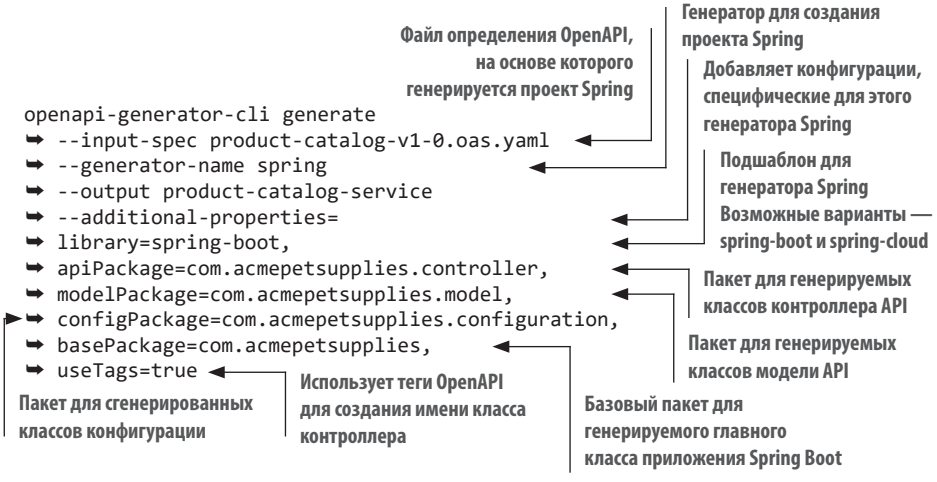
https://liveinternet.club

приложением Java Spring Boot, а объекты контроллера, логики маршрутизации, запроса и объекта должны генерироваться на основе файла определения OpenAPI, который служит источником истины, с использованием OpenAPI Generator.

ПРИМЕЧАНИЕ Читатели, знакомые с Java и Spring Boot, смогут создать код примера из каталога `chapter6` в Visual Studio Code (VS Code) или в любой Java IDE. Остальные могут заглянуть в каталог `chapter6-completed`, чтобы просмотреть готовый пример.

OpenAPI Generator (https://github.com/OpenAPITools/openapi-generator) — популярный инструмент с открытым исходным кодом для генерации серверного кода API, клиентского кода и документации на основе документов OpenAPI. Он подходит для многих языков программирования, фреймворков и форматов документации. Если вы выполнили инструкции в приложении Г, то у вас уже установлено Node-приложение командной строки `openapi-generator-cli` (https://github.com/OpenAPITools/openapi-generator-cli), которое загружает и запускает исполняемый файл OpenAPI Generator для Java. Перейдите в каталог `chapter6` и создайте в нем подкаталог `product-catalog-service`. Теперь выполните команду `openapi-generator-cli` из следующего листинга, чтобы создать свое приложение с объектами модели в пакете `com.acmepetsupplies.model` и заглушкой контроллера в пакете `com.acmepetsupplies.api`.

Листинг 6.1. Генерация приложения Spring Boot на основании определения OpenAPI



СОВЕТ Проследите за тем, чтобы между элементами, следующими после элемента `--additional-properties` в команде, не было пробелов. Кроме того, если вы не хотите вводить всю команду, можно запустить вспомогательный скрипт из папки `chapter6`, `listing-6-1-generate-spring-boot-app.sh` (или `listing-6-1-generate-spring-boot-app.bat` для Windows).

https://liveinternet.club

Взгляните на файлы, созданные в папке. Файл `README.md` содержит информацию, из которой следует, что проект был создан OpenAPI Generator. Файл `openapitools.json` содержит информацию о версии пакета OpenAPI Generator, который был загружен и выполнен `openapi-generator-cli` для генерации вашего проекта. Этот файл также может использоваться с целью определения сокращений параметров конфигурации для интерфейса командной строки (CLI), но эта тема выходит за рамки главы (о ней можно больше узнать по адресу https://github.com/OpenAPITools/openapi-generator-cli). Файл `.openapi-generator/FILES` содержит список всех файлов, сгенерированных при этом запуске. Файл `pom.xml` (POM — сокращение от Project Object Model) содержит информацию, необходимую инструментам сборки проекта для построения и настройки конфигурации проекта. OpenAPI Generator создал ваш проект как проект Maven (то есть проект настроен для построения средствами Maven), а Maven ищет информацию о сборке проекта в файле `pom.xml`. Файл `.openapi-generator-ignore` используется для задания имен и шаблонов файлов, которые не должны перезаписываться при повторных запусках генератора. Наконец, каталог `src` содержит исходный код Java и код тестов. Структура каталогов с созданными файлами выглядит так:

```
├── .openapi-generator
│   ├── FILES
│   └── VERSION
├── .openapi-generator-ignore
├── README.md
├── openapitools.json
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── acmepetsupplies
    │   │           ├── OpenApiGeneratorApplication.java
    │   │           ├── RFC3339DateFormat.java
    │   │           ├── configuration
    │   │           │   ├── HomeController.java
    │   │           │   └── SpringDocConfiguration.java
    │   │           ├── controller
    │   │           │   ├── ApiUtil.java
    │   │           │   ├── ProductsApi.java
    │   │           │   └── ProductsApiController.java
    │   │           └── model
    │   │               ├── Error.java
    │   │               └── Product.java
    │   └── resources
    │       ├── application.properties
    │       └── openapi.yaml
    └── test
        └── java
            └── com
                └── acmepetsupplies
                    └── OpenApiGeneratorApplicationTests.java
```

https://liveinternet.club

Мы рассмотрели структуру каталогов с файлами, созданными для проекта. Теперь обратимся к исходному коду в папке `src`. В каталоге `application` перейдите в папку `src/main/java/com/acmepetsupplies`. Файл `controller/ProductsApi.java` содержит интерфейс заглушки контроллера, содержащий информацию о маршрутизации в форме аннотации Java `@RequestMapping`. Эта аннотация сообщает фреймворку Spring Boot, что запросы для `GET /v1/catalog/products/{id}` должны направляться этому контроллеру, а контроллер возвращает ответ с типом содержимого `application/json`. Он содержит метод `viewProduct(id)`, имя которого базируется на значении `operationId` эндпоинта в файле определения API. Метод `viewProduct(id)` имеет реализацию по умолчанию, которая возвращает ответ, основанный на примере из файла OpenAPI `chapter6/product-catalog-v1-0.oas.yaml`. Аннотации `@Tag`, `@Operation` и `@ApiResponse` содержат метаданные документа из определения OpenAPI. Эти метаданные могут использоваться для повторной генерации определения OpenAPI из кода. Интерфейс `ProductsApi` приведен ниже:

```
@Generated(value = "org.openapitools.codegen.languages.SpringCodegen",
➥ date = "2022-12-31T13:20:55.039973Z[Europe/London]")
@Validated
@Tag(name = "Products", description = "Продукт – это единица товара
➥ для продажи в магазине.")
public interface ProductsApi {

    default Optional<NativeWebRequest> getRequest() {
        return Optional.empty();
    }

    @Operation(
        operationId = "viewProduct",
        summary = "Посмотреть подробности продукта",
        tags = { "Products" },
        responses = {
            @ApiResponse(responseCode = "200", description = "OK",
➥ content = {
                @Content(mediaType = "application/json",
➥ schema = @Schema(implementation =
➥ Product.class))
            }),
            …
        security = {
            @SecurityRequirement(name = "ApiKeyAuth")
        }
    )
    @RequestMapping(
        method = RequestMethod.GET,
        value = "/v1/catalog/products/{id}",
        produces = { "application/json" }
    )
    default ResponseEntity<Product> viewProduct(
        @Parameter(name = "id", description = "Идентификатор продукта",
➥ required = true) @PathVariable("id") UUID id
    ) {
```

Метаданные операций API документа OpenAPI

Метаданные ответов API документа OpenAPI

Направляет запросы для эндпоинта /v1/catalog/products/{id} методу viewProduct

https://liveinternet.club

```
        getRequest().ifPresent(request -> {
            for (MediaType mediaType:
➥ MediaType.parseMediaTypes(request.getHeader("Accept"))) {
                if (mediaType.isCompatibleWith(
➥ MediaType.valueOf("application/json"))) {
                    String exampleString = "{ \"keywords\" :
➥ [ \"keywords\", \"keywords\", \"keywords\", \"keywords\",
➥ \"keywords\" ], \"price\" : 100, \"name\" :
➥ \"Acme Uber Dog Rope Toy\", \"description\" :
➥ \"Acme Uber Dog Rope Toy не даст вашей собаке заскучать.
➥  Игрушку можно бросать и ловить в прыжке.\",
➥ \"id\" : \"dcd53ddb-8104-4e48-8cc0-5df1088c6113\" }";
                    ApiUtil.setExampleResponse(request,
➥ "application/json", exampleString);
                    break;
                }
            }
        });
        return new ResponseEntity<>(
➥ HttpStatus.NOT_IMPLEMENTED);
    }
}
```

Реализация по умолчанию возвращает пример ответа, если заголовок Accept содержит application/json

Возвращает ошибку 501 Not Implemented, если заголовок Accept отсутствует

Файл `controller/ProductsApiController.java` реализует интерфейс `ProductsApi`. В реальном приложении контроллер будет содержать некоторую логику или вызов объекта сервиса, содержащего бизнес-логику, и может читать информацию из базы данных. В данном примере мы переопределим метод `viewProduct(id)` и вернем пустышку (dummy object), чтобы продемонстрировать возможность возвращения нестандартного ответа.

Для этого сначала добавьте следующие директивы в начало файла `ProductsApiController.java`, сразу же после объявления пакета:

```
import io.swagger.v3.oas.annotations.Parameter;
import java.math.BigDecimal;
import java.util.Set;
import java.util.HashSet;
import java.util.Arrays;
```

Теперь вставьте код из листинга 6.2 в файл `ProductsApiController.java`. Он добавляет новый метод `viewProduct`, возвращающий в ответе жестко запрограммированный объект `Product`.

OpenAPI Generator создает в пакете модели два файла — `model/Product.java` и `model/Error.java`. Класс `Product.java` возвращается контроллером для успешных (2xx) ответов, а класс `Error.java` должен возвращаться кодом обработчика ошибки приложения. При сериализации в форме сообщения JSON класс `Error` соответствует схеме ответа об ошибке, определенной в документе OpenAPI. Впрочем, на данный момент еще нет кода обработки ошибок, который бы использовал класс `Error.java`. Чтобы это исправить, следует создать обработчик ошибок, который перехватывает любые исключения, выданные контроллером,

https://liveinternet.club

и возвращает экземпляр объектного типа `Error`. Для этого создайте файл `ApplicationExceptionHandler.java` в каталоге `src/main/java/com/acmesupplies/controller`, как показано в листинге 6.3.

Листинг 6.2. Добавление метода viewProduct в ProductApiController

Аннотация Swagger Parameter описывает параметр ID с информацией, которая может использоваться для генерации определения OpenAP на основе кода

Переопределяет метод viewProduct, определенный в интерфейсе ProductsApi

Создает жестко запрограммированный объект product, возвращаемый методом

Аннотация Spring Framework PathVariable означает, что параметр метода ID должен быть связан с шаблонной переменной Spring URI

```
    @Override
    public ResponseEntity<Product> viewProduct(
        @Parameter(name = "id", description = "Идентификатор продутка",
➥ required = true)
➥ @PathVariable("id") UUID id){
        Product product = new Product();
        product.setId(id);
        product.setName("Acme Uber Dog Rope Toy");
        product.setDescription
 ➥ ("Acme Uber Dog Rope Toy не даст заскучать вашей собаке.");
        product.setPrice(new BigDecimal(50));
        Set<String> keywords = new HashSet<>(Arrays.asList("rope", "toy",
 ➥ "dog"));
        product.setKeywords(keywords);
        return new ResponseEntity<Product>(product, HttpStatus.OK);
    }
}
```

Листинг 6.3. Общий код обработки ошибок

```
package com.acmepetsupplies.controller;

import com.acmepetsupplies.model.Error;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.lang.Nullable;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
➥ ResponseEntityExceptionHandler;

import java.util.UUID;
```

Перехват исключений от контроллера и применение логики обработки ошибок

```
@RestControllerAdvice
public class ApplicationExceptionHandler extends
ResponseEntityExceptionHandler {
    @Override
    protected ResponseEntity<Object> handleExceptionInternal(
            Exception ex, @Nullable Object body, HttpHeaders headers,
```

https://liveinternet.club

```
➥ HttpStatus status, WebRequest request) {
        Error apiError = new Error();
        apiError.setId(UUID.randomUUID());
        apiError.setStatus(status.value());
        switch (status.series().value()) {
            case 4: apiError.setCode("validation." +
➥ status.series().name().toLowerCase());
                    break;
            case 5: apiError.setCode("error." +
➥ status.series().name().toLowerCase());
                    break;
            default:
➥ apiError.setCode(status.series().name().toLowerCase());
                    break;
        }
        apiError.setTitle(status.getReasonPhrase());
        switch (status.value()) {
            case 400: apiError.setDetail("Неверный запрос. Проверьте, что
➥ запрос допустим.");
                    break;
            case 404:
                    apiError.setDetail("Ресурс не найден. Проверьте
➥ путь и идентификатор ресурса в запросе.");
                    break;
            case 429: apiError.setDetail("Слишком много запросов. Превышен лимит
➥ запросов в единицу времени. Попробуйте позже.");
                    break;
            default: apiError.setDetail(status.getReasonPhrase());
                    break;
        }
        return new ResponseEntity<>(apiError, headers, status);
    }
}
```

Метод настройки тела ответа для всех типов исключений

В зависимости от статуса HTTP задается подробная информация объекта ошибки

Вы сгенерировали проект Spring Boot, создали контроллер и написали общий код обработки ошибок. Теперь необходимо сообщить Spring Boot, как автоматически настроить и запустить проект. Запустите приложение из командной строки командой `mvn spring-boot:run`. Команда запускает приложение Spring Boot и открывает доступ к нему на `localhost:8080`. В предпочитаемом клиенте HTTP отправьте запрос к приложению, используя любое значение UUID (Universally Unique Identifier) как идентификатор продукта; вы увидите продукт, возвращаемый в ответе. В следующем примере я использую `curl` и форматирую ответ в `jq`:

```
curl --silent --request GET
➥ http://localhost:8080/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b80fc | jq .
```

> **СОВЕТ** Для пользователей Windows приведенная выше команда работает в командной строке Cmd.exe, но в PowerShell curl является встроенным псевдонимом для командлета InvokeWebRequest. Таким образом, чтобы выполнить эту команду в PowerShell, следует заменить curl на curl.exe.

https://liveinternet.club

Команда выдает следующий ответ:

```
{
  "id": "612b4280-b5c0-4ad5-bce7-ede7ab2b80fc",
  "name": "Acme Uber Dog Rope Toy",
  "description": "Acme Uber Dog Rope Toy не даст
➥ вашей собаке заскучать.",
  "price": 50,
  "keywords": [
    "rope",
    "dog",
    "toy"
  ]
}
```

Вы получили успешный ответ, и это замечательно. Но что произойдет, если отправить запрос с идентификатором продукта, который не является UUID? Проверит ли приложение входной параметр? Вспомните, что вы не написали никакой логики проверки. Чтобы узнать, что произойдет, отправьте запрос с простой строкой, которая не является UUID:

```
curl --silent --request GET
➥ http://localhost:8080/v1/catalog/products/not-a-uuid | jq
{
  "id": "d207deea-cef0-4598-ae86-ebb56340b4ac",
  "status": 400,
  "code": "validation.client_error",
  "title": "Bad Request",
  "detail": "Неверный запрос. Проверьте, что запрос допустим."
}
```

Приложение проверяет ваш ввод и возвращает ответ, соответствующий схеме ошибки из определения OpenAPI. Фреймворк приложений Spring Boot использует аннотации валидации объектов контроллера и модели для проверки пользовательского ввода.

Поздравляем! Вы успешно сгенерировали свой сервис API на основе источника истины – определения OpenAPI. Работу сервиса можно прервать комбинацией клавиш Ctrl+C, но пока пусть он работает. В следующем разделе я покажу, как развернуть сервис API на API-шлюзе.

6.2.2. Развертывание на шлюзе

Представим, что на следующем шаге ваш тимлид требует развернуть сервис API на API-шлюзе (шаг 3 на рис. 6.5). API-шлюз является обратным прокси-сервером и центральной точкой входа, которая направляет внешние запросы сервисам API на бэкенде. Кроме маршрутизации, API-шлюз может предоставлять другие сквозные функции: аутентификацию, преобразование протоколов, ограничение частоты и мониторинг. API-шлюзы позволяют отделить интерфейс внешнего клиента от внутреннего бэкенд-сервиса.

https://liveinternet.club

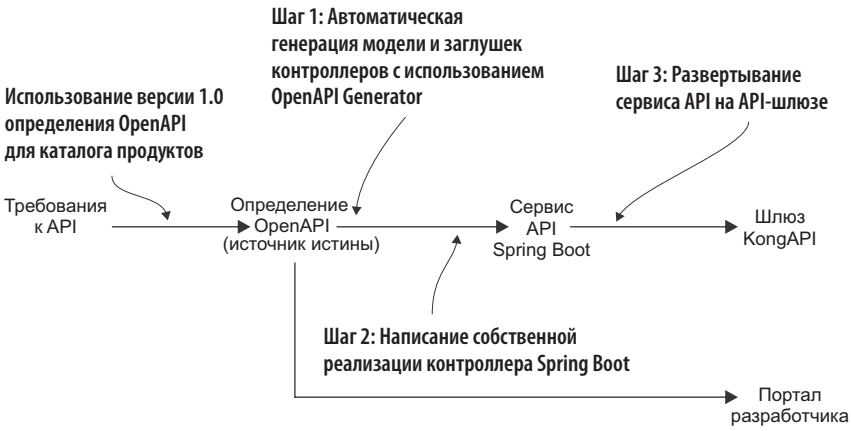


Рис. 6.5. На следующем шаге сервис API развертывается на API-шлюзе

В этом разделе описан процесс развертывания API в Kong — популярном API-шлюзе с открытым кодом. Хотя в этом примере используется шлюз Kong, еще раз подчеркну, что я не пытаюсь научить вас использованию конкретной технологии. Цель в том, чтобы показать, что для общедоступного API, который здесь строится, документация API должна отражать поведение всех компонентов, которые вносят свой вклад во внешне наблюдаемое поведение API. К этой категории относится как сервис API, так и API-шлюз. Как будет показано ниже, API-шлюз может вернуть ответ, о котором сервису API ничего не известно, но этот факт все равно должен быть отражен в документации API.

Когда вы развертываете сервис API на шлюзе, вы создаете маршрут между внешним запросом и сервисом API. По умолчанию Kong прослушивает общедоступный трафик HTTP на порту 8080. Настройте Kong так, чтобы он сопоставлял и перенаправлял любые запросы к `GET /v1/catalog/products/(id)` с заголовком хоста `api.acme-pet-supplies.com` на ваш сервис каталога продуктов, работающий на порту 8080. Для этого обновите файл `chapter6/kong/kong.yaml`, как показано в листинге 6.4.

Листинг 6.4. Декларативная конфигурация API-шлюза Kong

```
_format_version: "1.1"
services:
  - name: Product_Catalog_API
    protocol: http
    host: localhost
    port: 8080
    path:  /
    routes:
      - tags:
          - Product Catalog
```

https://liveinternet.club

```
        name: viewProduct
        methods:
          - GET
        hosts:
          - api.acme-pet-supplies.com
        paths:
          - /v1/catalog/products/(.+)$
        strip_path: false
    plugins:
      - name: key-auth
        config:
          key_names:
          - x-api-key
consumers:
- username: my_api_consumer
  keyauth_credentials:
  - key: my_secret_api_key
```

Запросы к GET /v1/catalog/products/{id} с заголовком хоста, содержащим api.acme-pet-supplies.com, перенаправляются на бэкенд-сервис API на localhost:8080

Плагин key-auth обеспечивает аутентификацию ключа API

ПРИМЕЧАНИЕ Пользователям, запускающим Kong в Docker (например, пользователям Windows), необходимо заменить значение `host` в листинге 6.4 на `host.docker.internal`. Это специальное имя доменной системы имен (DNS), которое преобразуется во внутренний IP-адрес, используемый хост-компьютером Windows. Это позволяет экземпляру Kong, выполняемому в Docker, направлять запрос сервису API каталога продуктов в бэкенде, выполняемому на хосте Windows.

Файл `kong.yaml` содержит декларативную конфигурацию для маршрутизации запросов шлюзу Kong. С такой конфигурацией все запросы к шлюзу API должны предоставлять заголовок `Host` со значением `api.acme-pet-supplies.com`, чтобы маршрутизироваться к сервису API на бэкенде.

Секция `securityScheme` в определении OpenAPI указывает, что для защиты API используется ключ API. API-шлюз — стандартное место для реализации аутентификации ключей API. Ключ API используется для идентификации приложения (например, потребителя API), вызывающего API. Конфигурация в листинге 6.4 настраивает Kong так, чтобы потребители API отправляли ключ API, для чего активируется плагин Kong `key-auth`. В данном случае для выдачи успешного запроса к серверу потребитель API должен отправить заголовок `x-api-key` со значением `my_secret_api`.

Перейдите в каталог `chapter6/kong` и запустите Kong командой `kong start -c kong.conf`. (Пользователи Windows используют для запуска Kong скрипт `run_kong_docker.bat` из папки `chapter6/kong`. Если вы работаете в операционной системе из семейства Unix, также можно использовать вспомогательные скрипты `start_kong.sh` и `stop_kong.sh` для запуска и остановки Kong.) Затем в предпочитаемом клиенте API отправьте запрос к API-шлюзу и передайте ключ API в запросе, задавая заголовки `Host` и `x-api-key`. В `curl` это делается так:

```
curl --silent --header "Host: api.acme-pet-supplies.com"
➥ --header "x-api-key: my_secret_api_key"
➥ --request GET http://localhost:8000/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b80fc | jq .
```

https://liveinternet.club

На консоли должен появиться ответ с продуктом. Поздравляю! Вы успешно развернули API на шлюзе и отправили запрос к защищенному общедоступному API.

6.2.3. Публикация на портале разработчика

Представьте, что ваш тимлид просит опубликовать на портале разработчика определение API, служащее источником истины, как документацию API (рис. 6.6).

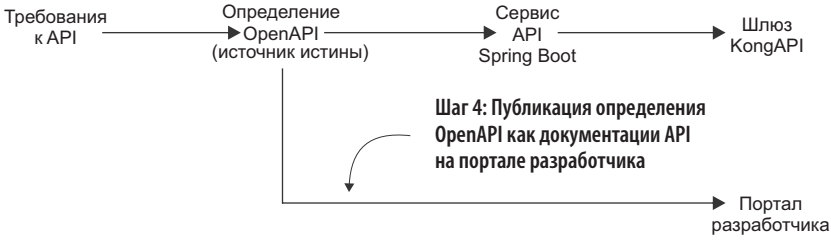


Рис. 6.6. Публикация определения API как документации API на портале разработчика

Портал разработчика предоставляет авторитетную информацию, к которой обращаются потребители API, когда хотят узнать, как пользоваться общедоступным API. На порталах разработки также размещаются руководства пользователя API и учебники, которые помогают разработчикам быстрее освоить использование API. Также имеется документация API со списком операций, поддерживаемых API. На практике можно отправить свою документацию API в сервис портала разработчика, использовать портал разработчика, который прилагается к платформе API-шлюза, и даже создать собственный портал. Для сценария из нашего примера создайте в каталоге `chapter6` папку с именем `dev-portal`. Используйте ее для хранения и предоставления доступа к созданной вами документации API.

Сгенерируйте документацию API на основе определения API при помощи Redocly CLI. Этот сервис генерирует документацию API в визуальном интерфейсе, где левая панель содержит строку поиска и меню навигации. Если вы выполнили инструкции из приложения Г, Redocly CLI уже установлен на вашем компьютере. В каталоге `chapter6` выполните следующую команду, чтобы сгенерировать документацию API в форме статического HTML-файла без зависимостей:

```
redocly build-docs product-catalog-v1-0.oas.yaml
➥ --output dev-portal/reference.html
```

В папке `dev-portal` запустите HTTP-сервер, который будет предоставлять HTML-файл, командой `http-server --port 9000`. В браузере перейдите по адресу http://localhost:9000/reference.html, чтобы просмотреть сгенерированную документацию API.

https://liveinternet.club

Поздравляю, вы успешно опубликовали документацию API! Сейчас она точно документирует успешный ответ 200 эндпоинта `GET /v1/catalog/products/{id}` вашего API. Но что, если поведение API изменится и определение станет неверным? Вспомните, что в такой ситуации возможна рассинхронизация документации API с кодом. На следующем шаге вы узнаете, как синхронизировать изменения в коде и документации API при обновлении источника истины.

6.2.4. Обновление источника истины

Допустим, вы получили изменение в требованиях, которое приведет к изменению источника истины в API. Бизнес-аналитик вашей команды сообщает, что по новым требованиям должны возвращаться два новых обязательных поля, `reviewRating` и `numberofReviews`; оба содержат целые числа. Следующий фрагмент из схемы Products в обновленном определении API из файла `product-catalog-v1-1.oas.yaml` включает новые обязательные атрибуты в ответ эндпоинта:

```
reviewRating:
  maximum: 5
  minimum: 1
  type: integer
  description: Средняя оценка товара по отзывам.
  format: int32
  example: 5
numberOfReviews:
  type: integer
  description: Количество отзывов о товаре.
  format: int32
  example: 10
```

Так как определение OpenAPI является источником истины об операциях API, оно обновляется в файле `product-catalog-v1-1.oas.yaml` в соответствии с новыми требованиями. Теперь необходимо заново сгенерировать объекты модели ответа в сервисе API, чтобы они включали два новых поля, и заполнить их. Чтобы документация API соответствовала поведению сервиса API, необходимо обновить ее на основании источника истины.

Код сервиса API можно сгенерировать по обновленному файлу `product-catalog-v1-1.oas.yaml`. При этом не должны быть перезаписаны изменения, добавленные в файл `ProductsApiController.java`. Для этого добавьте `**ProductsApiController.java` в файл `product-catalog-service/.openapi-generator-ignore`. Затем перейдите в папку `chapter6` и выполните команду из листинга 6.5, чтобы сгенерировать код с использованием файла `product-catalog-v1-1.oas.yaml`.

СОВЕТ Также можно выполнить вспомогательный скрипт `listing-6-5-generate-spring-boot-app.sh` или `listing-6-5-generate-spring-boot-app.bat` в зависимости от вашей операционной системы.

https://liveinternet.club

Листинг 6.5. Повторная генерация сервиса Spring Boot

```
openapi-generator-cli generate
➥ --input-spec product-catalog-v1-1.oas.yaml
➥ --generator-name spring
➥ --ignore-file-override
➥ product-catalog-service/.openapi-generator-ignore
➥ --additional-properties=
➥ library=spring-boot,
➥ apiPackage=com.acmepetsupplies.controller,
➥ modelPackage=com.acmepetsupplies.model,
➥ configPackage=com.acmepetsupplies.configuration,
➥ basePackage=com.acmepetsupplies,
➥ useTags=true
```

Использует обновленное определение (источник истины)

Задает местоположение ignore-файла, чтобы предотвратить перезапись ProductsApiController.java

Обратите внимание: файл `ProductsApiController.java` не обновляется, но заново сгенерированный файл `Product.java` теперь содержит геттеры и сеттеры для `reviewRating` и `numberOfReviews`.

Использование Maven-плагина OpenAPI Generator

Вместо openapi-generator-cli Java-разработчики могут предпочесть создание цели Maven (то есть исполняемого плагина) для повторной генерации проекта. openapi-generator-maven-plugin позволяет сгенерировать код серверной заглушки из проекта Maven. Такая возможность полезна, так как для генерации кода можно воспользоваться командами Maven. Включите следующий фрагмент в раздел `plugins` файла `pom.xml`, чтобы сгенерировать код серверной заглушки в каталоге `target`:

```
<plugins>
    <plugin>
        <groupId>org.openapitools</groupId>
        <artifactId>openapi-generator-maven-plugin</artifactId>
        <version>6.2.1</version>
        <executions>
            <execution>
                <goals>
                    <goal>generate</goal>
                </goals>
                <id>buildApi</id>
                <configuration>
                    <inputSpec>
   ➥ ${project.basedir}/../product-catalog-v1-1.oas.yaml
   ➥ </inputSpec>
                    <generatorName>spring</generatorName>
                    <library>spring-boot</library>
                    <ignoreFileOverride>
   ➥ ${project.basedir}/.openapi-generator-ignore
   ➥ </ignoreFileOverride>

            <modelPackage>com.acmepetsupplies.model</modelPackage>

            <apiPackage>com.acmepetsupplies.controller</apiPackage>
                <invokerPackage>com.acmepetsupplies</invokerPackage>
```

Ссылается на местоположение файла определения API

Задает ignore-файл генератора

https://liveinternet.club

```
                    <configOptions>
                        <useTags>true</useTags>
<configPackage>com.acmepetsupplies.configuration</configPackage>
                    </configOptions>
                </configuration>
            </execution>
        </executions>
    </plugin>
…
```

Затем создайте цель — плагин Maven для копирования сгенерированных файлов в папку `src/`. Для этого можно воспользоваться плагином Maven AntRun:

```
        <plugin>
            <artifactId>maven-antrun-plugin</artifactId>
            <version>3.0.0</version>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <configuration>
                        <target>
                            <move todir="src/main">
                                <fileset dir=
➥ "target/generated-sources/openapi/src/main"/>
                            </move>
                        </target>
                    </configuration>
                    <goals>
                        <goal>run</goal>
                    </goals>
                </execution>
            </executions>
    </plugin>
…
```

Плагин Maven AntRun для копирования сгенерированных исходных файлов из каталога target в src/main/

Теперь выполните команду `mvn clean compile`, чтобы очистить каталог `target`, сгенерировать и перезаписать файлы как часть этапа компиляции. Обратите внимание: в отличие от других плагинов Maven, прямой запуск командой `mvn openapi-generator:generate` не сработает и произойдет ошибка.

Обновите `ProductApiController.java`, чтобы полям `reviewRating` и `numberOfReviews` возвращаемого продукта присваивались любые целочисленные значения. В листинге 6.6 показано, как должен выглядеть метод `viewProduct` класса `ProductApiController`.

Листинг 6.6. Обновление viewProduct в ProductApiController

```
@Override
public ResponseEntity<Product> viewProduct(
        @Parameter(name = "id", description = "Идентификатор продукта",
➥ required = true) @PathVariable("id") UUID id) {
    Product product = new Product();
```

https://liveinternet.club

```
    product.setId(id);
    product.setName("Acme Uber Dog Rope Toy");
    product.setDescription(
➥ "Acme Uber Dog Rope Toy не даст заскучать вашей собаке.");
    product.setPrice(new BigDecimal(50));
    Set<String> keywords =
➥ new HashSet<>(Arrays.asList("rope", "toy", "dog"));
    product.setKeywords(keywords);
    product.setNumberOfReviews(11);
    product.setReviewRating(3);
    return new ResponseEntity<Product>(product, HttpStatus.OK);
}
```

Задает количество отзывов и оценку товара по отзывам

Остановите приложение (нажмите Ctrl+C, чтобы остановить работающий экземпляр сервера) и перезапустите его командой `mvn spring-boot:run`, чтобы подключить последние изменения. Так как приложение уже маршрутизируется через API-шлюз, вы сможете отправить запрос к шлюзу при помощи клиента API и увидеть обновленный ответ. Для этого используется следующая команда:

```
curl --silent --header "Host: api.acme-pet-supplies.com"
➥ --header "x-api-key: my_secret_api_key"  --request GET
➥ http://localhost:8000/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b80fc | jq .
```

Вы увидите ответ с двумя новыми полями:

```
{
    "id": "612b4280-b5c0-4ad5-bce7-ede7ab2b80fc",
    "name": "Acme Uber Dog Rope Toy",
    "description": "Acme Uber Dog Rope Toy не даст заскучать
➥ вашей собаке.",
    "price": 50,
    "keywords": [
        "rope",
        "dog",
        "toy"
    ],
    "reviewRating": 3,
    "numberOfReviews": 11
}
```

Обратите внимание на новые поля в ответе

Теперь ваш API функционирует в соответствии с новыми требованиями, а к обновленному эндпоинту можно обратиться из API-шлюза. Однако документация API на портале разработчика теперь не синхронизирована, и ее нужно обновить.

6.2.5. Повторная публикация определения API на портале

Следующий шаг – повторно сгенерировать документацию API на основе источника истины, чтобы она соответствовала поведению API. Сначала остановите работающий экземпляр HTTP-сервера комбинацией Ctrl+C. Затем выполните

https://liveinternet.club

следующие команды из каталога `chapter6`, чтобы удалить старую документацию API и сгенерировать новую на основе обновленной версии определения API:

```
rm  dev-portal/reference.html
redocly build-docs product-catalog-v1-1.oas.yaml
➥ --output dev-portal/reference.html
```

Перейдите в папку `dev-portal` и запустите HTTP-сервер командой `http-server -port 9000`. Откройте страницу http://localhost:9000/reference.html, чтобы увидеть обновленную документацию API. В ней присутствуют два новых атрибута (`reviewRating` и `numberOfReviews`) в ответе 200 эндпоинта `viewProduct`, что соответствует поведению API.

6.2.6. Генерация клиентских SDK

В рассмотренном примере кое-чего не хватает: мы не сгенерировали клиентский SDK. Допустим, вам нужно сгенерировать Java SDK для API каталога продуктов из определения. Клиентский SDK для API предоставляет библиотеку, которая используется потребителями API для отправки запросов к API вместо включения прямых HTTP-запросов в код. SDK инкапсулирует нижележащий HTTP-клиент для направления HTTP-запросов к серверу API (рис. 6.7).



Рис. 6.7. Клиентский SDK для API — библиотека, инкапсулирующая HTTP-клиент, который совершает низкоуровневые вызовы к API

OpenAPI Generator позволяет сгенерировать клиентскую библиотеку по той же схеме, по которой генерируется серверный код, — с указанием имени используемого генератора и параметров, управляющих этим генератором. В зависимости от заданного генератора также есть возможность задать нижележащий HTTP-клиент, который будет использоваться для отправки HTTP-запросов.

> **СОВЕТ** Чтобы просмотреть список генераторов, доступных в OpenAPI Generator, выполните команду `openapi-generator-cli list` в командной строке или перейдите по адресу https://openapi-generator.tech/docs/generators.

Чтобы увидеть, как это делается, создайте папку `java-client` в каталоге `chapter6` и выполните код из листинга 6.7, генерирующий проект клиентского SDK для API на Java.

https://liveinternet.club

Листинг 6.7. Генерация клиентского SDK для API на Java

```
openapi-generator-cli generate
➥ --input-spec ../product-catalog-v1-1.oas.yaml
➥ --generator-name java
➥ --output java-client
➥ --additional-properties=
➥ library=okhttp-gson,
➥ invokerPackage=com.acmepetsupplies.client,
➥ apiPackage=com.acmepetsupplies.api,
➥ modelPackage=com.acmepetsupplies.model,
➥ groupId=com.acmepetsupplies,
➥ artifactId=acmepetsupplies-java-client
```

Имя генератора клиентских SDK

Используемый шаблон библиотеки: клиент HTTP okhttp с процессором JSON gson

Значение Maven groupId, используемое для идентификации проекта

Значение Maven artifactId, определяющее имя создаваемого JAR-файла

СОВЕТ Также можно выполнить вспомогательный скрипт `listing-6-7-generate-java-client.sh` или `listing-6-7-generate-java-client.bat` из каталога `chapter6` в зависимости от вашей операционной системы.

OpenAPI Generator создал класс `chapter6/java-client/src/main/java/com/acmepetsupplies/api/ProductsApi.java` с методом `viewProduct(id)`, использующим нижележащий клиент `OkHttp` для совершения вызовов к вашему API. Соответствующий фрагмент начинается со строки 70 в файле:

```
public Product viewProduct(UUID id) throws ApiException {
    ApiResponse<Product> localVarResp = viewProductWithHttpInfo(id);
    return localVarResp.getData();
}
```

Чтобы протестировать эту библиотеку, создайте в проекте тестовый класс, который отправляет запрос и выводит ответ без каких-либо условий. Откройте каталог `java-client` в Visual Studio Code (VS Code) или предпочитаемом редакторе Java, чтобы просмотреть сгенерированные файлы. Откройте файл `src/test/java/com/acmepetsupplies/api/ProductsApiTest.java` и обновите его, как показано в листинге 6.8. (Не забудьте удалить аннотацию `@Disabled` в классе `ProductsApiTest`.)

Листинг 6.8. Использование SDK

```
public class ProductsApiTest {
    @Test
    public void viewProductTest() throws ApiException {
        ProductsApi api = new ProductsApi();
        api.getApiClient().setBasePath(
➥ "http://localhost:8000");
        api.getApiClient().addDefaultHeader(
➥ "Host", "api.acme-pet-supplies.com");
        api.getApiClient().setApiKey(
➥ "my_secret_api_key");
```

Инициализирует класс ProductsApi, инкапсулирующий клиент API

Использует клиент API для задания значений заголовка и ключа API

https://liveinternet.club

```
        UUID id = UUID.randomUUID();
        Product response = api.viewProduct(id);   <-- Вызывает операцию API для отправки запроса HTTP
        System.out.println(response);
    }
}
```

Запустите тест. Это можно сделать из командной строки командой `mvn test`. На консоли должен появиться вывод с ответом API.

Теперь нужно упаковать библиотеку в файл JAR, чтобы потребители API могли добавить ее в виде зависимости в свои проекты Java. Для этого выполните команду `mvn install`. Она создает библиотеку `target/acmepet-supplies-java-client-1.0.jar`, которая может использоваться в разных проектах. Копия также добавляется в локальный репозиторий Maven на вашем компьютере, чтобы библиотека была доступна в других созданных вами проектах Maven. Также можно опубликовать библиотеку во внешнем репозитории Maven, но эта тема выходит за рамки книги.

Как и при генерации серверного кода, клиентский SDK для API генерируется быстро, и вы можете заново это делать при каждом обновлении определения OpenAPI. Помимо клиента `OkHttp`, этот генератор поддерживает другие HTTP-клиенты для Java, включая OpenFeign, Spring RestTemplate, Jersey и т. д.

Если вы создадите общедоступный API и свое определение OpenAPI, но не предложите SDK для языков программирования, некоторые потребители API могут сгенерировать SDK самостоятельно. А значит, будет полезно сгенерировать несколько внутренних SDK на основе вашего определения OpenAPI для разных языков программирования, и убедиться, что при этом создаются именно те классы и методы, которые вы ожидали. Например, имя `operationId` в OpenAPI может начинаться с цифры, но имена методов Java не могут. Следовательно, при генерировании клиента Java OpenAPI Generator должен добавить к имени метода префикс, чтобы оно стало действительным.

6.2.7. Преимущества генерации на основе источника истины

Из рассмотренного примера видно, что изменения в требованиях API сначала отражаются в источнике истины (определении OpenAPI), на основе которого затем генерируются артефакты — серверный код, клиентские SDK и документация. В следующей главе вы увидите, что тесты также могжно генерировать на основе источника истины. В зависимости от задействованного в проекте количества команд разработчиков такое использование источника истины может естественно подходить для параллелизации. После того как содержимое источника будет согласовано, разные команды, работающие над серверным кодом, клиентским кодом или документацией, могут сгенерировать нужные им артефакты, исходя из согласованного определения API (рис. 6.8).

https://liveinternet.club

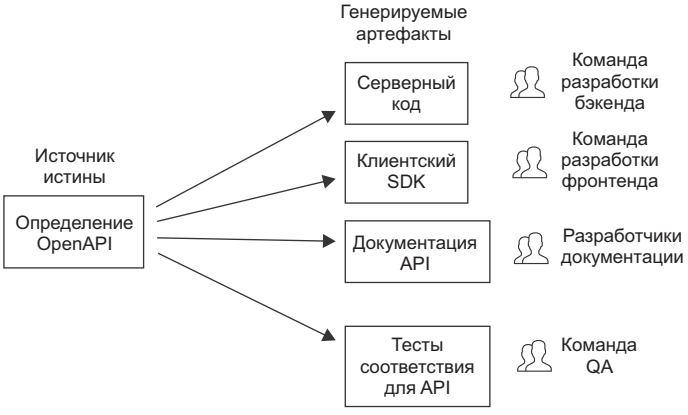


Рис. 6.8. Использование определения OpenAPI в качестве источника истины для генерации артефактов

В том, что касается соответствия кода документации API, еще одним преимуществом генерации кода на основе источника истины является скорость. Генерация выполняется быстро и с относительно низкими затратами на обслуживание по сравнению с тестами схем, проверяющими соответствие кода документации. Писать тесты или заниматься их обслуживанием не нужно. Можно быстро заново сгенерировать код заглушки сервиса API из командной строки. Такая скорость хорошо сочетается с гибкостью, необходимой для итеративного проектирования API. На практике проектирование API редко бывает линейным процессом с четкими этапами проектирования и сборки. Даже на этапе сборки некоторые параметры API могут обновляться, когда команда получает новую информацию о требованиях, разрабатываемой системе и технических ограничениях. Когда это происходит, при помощи генерации можно легко воссоздать все артефакты в соответствии с обновленным источником истины и продолжить разработку.

Я придерживаюсь мнения, что, используя подход на основе источника истины, полезно иметь два представления определения API — «как будет» и «как есть». Представление «как будет» ориентировано на будущий дизайн определения API с изменениями, которые отражают то, что должен делать API, но еще не сгенерированы в коде. Представление «как есть» — это определение API, которое точно отражает, что реализовано в API на данный момент. Например, в только что рассмотренном примере изменение с добавлением двух новых полей в ответ API было сначала применено к файлу определения API `product-catalog-v1-1.oas` (представлению «как будет»). На этот момент, поскольку изменение еще не было реализовано в сервисе API, представление «как будет» отличается наличием этих двух полей от файла `product-catalog-v1-0.oas.yaml` (представление «как есть»), который отражает текущее состояние реализации. Конечно, после того как код сервиса API будет заново сгенерирован на основе

https://liveinternet.club

файла `product-catalog-v1-1.oas.yaml`, `product-catalog-v1-1.oas.yaml` станет представлением «как есть».

Используя подход design-first («сначала проектирование», или «сначала контракт», или «сначала спецификация» — встречаются разные названия) для создания определения API, служащего источником истины, до начала разработки API, важно заметить, что разным участникам команды эти представления («как будет» и «как есть») понадобятся на разных стадиях жизненного цикла продукта API. На этапе проектирования для проектировщиков API и команды управления API важно представление «как будет». Но по мере того, как разработка API переходит от сборки к фазе публикации, возникает интерес к фазе «как есть», потому что именно эта информация будет опубликована на портале разработчика для потребителей API. Некоторые команды считают, что представление «как есть» полезно генерировать по коду, так как в этом случае оно наиболее точно отражает реализацию.

6.2.8. Обработка ответов от шлюза

При генерации серверного кода API на основе определения API возникают некоторые затруднения. Одно из них заключается в том, что ответы API могут поступать не от сервиса API, а от компонента, размещенного перед ним. Следовательно, нужно настроить эти компоненты для генерации ответов в нужном формате. Например, API-шлюз может отвечать за аутентификацию ключа API потребителя и отправлять ответ для ошибок аутентификации ключа (ошибка 401). Протестируйте этот сценарий в своем приложении, отправив запрос без ключа API. Выполните следующую команду:

```
Curl –include –header "Host: api.acme-pet-supplies.com"
➥ --request GET http://localhost:8000/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b80fc
```

Ответ, возвращенный шлюзом, должен выглядеть примерно так:

```
HTTP/1.1 401 Unauthorized
Date: Thu, 15 Dec 2022 06:57:52 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
WWW-Authenticate: Key realm="kong"
Content-Length: 45
X-Kong-Response-Latency: 0
Server: kong/3.0.0
{
  "message":"Ключ API в запросе не найден"
}
```

В такой ситуации у вас два варианта: (1) обновить ответ 401 в определении API в соответствии с этой структурой ответного сообщения и (2) настроить Kong так, чтобы возвращаемый результат соответствовал структуре объекта ответа об

https://liveinternet.club

ошибке (с полями `id`, `status`, `code`, `title` и `detail`). На момент написания книги для этого требуется написать специальный плагин Kong.

Обеспечение соответствия за счет генерации сервиса API по внешнему определению API наиболее эффективно, когда API-шлюз выполняет с запросом и ответом сервиса API минимум возможных преобразований. Рассмотрим сценарий, в котором по какой-то причине (возможно, для поддержки унаследованной функциональности) API-шлюз должен преобразовать полученный запрос, прежде чем перенаправлять его сервису API. Это может быть преобразование пути или строки запроса, удаление или добавление заголовков, удаление или замена атрибутов в теле запроса и т. д. В API-шлюзе Kong это делается при помощи плагина Request Transformer. Определение API, используемое для генерации документации API, должно точно представлять внешнее поведение API, но из-за значительного объема логики преобразования в шлюзе оно не может использоваться для генерации кода сервиса API. В микросервисной архитектуре могут быть и другие компоненты, обрабатывающие запросы до того, как они достигнут API-сервиса. Например, компонент управления политикой — такой, как OPA (Open Policy Agent), — может перехватывать запросы и возвращать ответы об ошибках авторизации. Поскольку компоненты, внешние по отношению к сервису API, могут добавлять логику в поток обработки запроса, для обеспечения соответствия документации API желательно дополнить генерацию API тестированием соответствия API (которое рассматривается в следующей главе).

6.2.9. Настройка генерации кода

OpenAPI Generator поддерживает более 40 языков программирования. В нашем примере OpenAPI Generator запускался из командной строки командой `openapi-generator-cli`. Но у OpenAPI Generator имеются плагины для интеграции с такими средствами автоматизации сборки, как Maven, Gradle (для Java), SBT (для Scala), Cake (для C#), Bazel (мультиязыковой инструмент сборки) и многими другими.

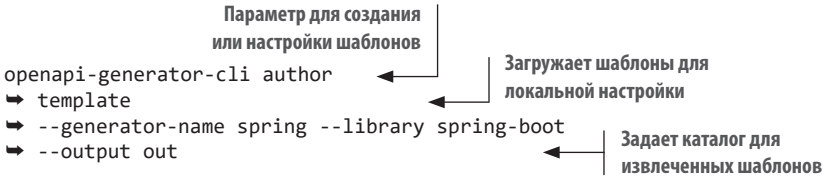
> **ПРИМЕЧАНИЕ** В этом разделе мы рассматривали OpenAPI Generator, но существуют и другие открытые генераторы кода OpenAPI. Например, NSwag (https://github.com/RicoSuter/NSwag) позволяет генерировать заглушки контроллера на C# по спецификации OpenAPI. OpenAPI Client and Server Code Generator (https://github.com/deepmap/oapi-codegen) позволяет генерировать код для приложений Go.

Если ни один из генераторов, входящих в поставку OpenAPI Generator, вам не подходит, вы можете расширить один из них: определить собственную логику генерации кода. Есть три способа сделать это. Если вам нужно лишь слегка изменить шаблон, используемый встроенным генератором, его можно извлечь и изменить. Допустим, вам необходимо изменить шаблон для генератора Spring Boot и удалить из него аннотацию `@Generated(value = "org.openapitools.codegen.languages.SpringCodegen",date ..)`, которая присутствует в генерируемых контроллерах. Для этого в каталоге `chapter6` выполните команду из листинга 6.9,

https://liveinternet.club

чтобы извлечь шаблоны для библиотеки `spring-boot` генератора Spring Boot, который вы хотите изменить. Извлеченные шаблоны появляются в каталоге `out`.

Листинг 6.9. Генерация шаблонов для генератора Spring Boot



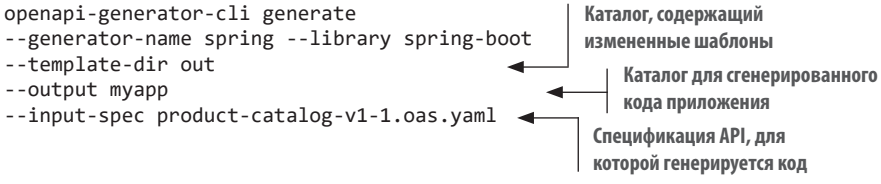

```
openapi-generator-cli author
➥ template
➥ --generator-name spring --library spring-boot
➥ --output out
```

СОВЕТ Также можно выполнить вспомогательный скрипт `listing-6-9-generate-templates.sh` или `listing-6-9-generate-templates.bat` из каталога `chapter6` в зависимости от вашей операционной системы.

Для шаблонов генерации кода OpenAPI Generator использует язык шаблонов Mustache (http://mustache.github.io, в реализации JMustache https://github.com/samskivert/jmustache). Отредактируйте шаблон `template-test/out/apiController.mustache` и удалите строку `{{>generatedAnnotation}}`.

Теперь сгенерируйте проект, запустив OpenAPI Generator со ссылкой на измененный каталог шаблонов, как показано в листинге 6.10. Команда генерирует новые файлы проекта в каталоге `myapp`.

Листинг 6.10. Новый проект с использованием измененных шаблонов Spring Boot




```
openapi-generator-cli generate
--generator-name spring --library spring-boot
--template-dir out
--output myapp
--input-spec product-catalog-v1-1.oas.yaml
```

СОВЕТ Также можно выполнить вспомогательный скрипт `listing-6-10-generate-from-custom-template.sh` или `listing-6-10-generate-from-custom-template.bat` из каталога `chapter6` в зависимости от вашей операционной системы.

Просмотрите сгенерированный файл `myapp/src/main/java/org/openapitools/api/V1ApiController.java`. Обратите внимание: аннотация `@Generated` в нем отсутствует.

Помимо встроенных шаблонов, поставляемых с генераторами, существуют два других способа настройки генерации кода: создание шаблонов, определяемых пользователями, и определение новых генераторов. Подробные описания настройки выходят за рамки книги, но дополнительную информацию о ней можно найти по адресу https://openapi-generator.tech/docs/customization.

https://liveinternet.club

6.3. Генерация OpenAPI из кода

Вместо того чтобы генерировать код на основе определения OpenAPI, можно пойти другим путем. Чтобы обеспечить соответствие документации коду, можно сгенерировать определение OpenAPI на основе кода, а затем создать артефакты (например, клиентские SDK и документацию API) исходя из сгенерированного определения. При таком подходе «сначала код» вы сначала вносите изменения, которые должны быть отражены в документации API, в код приложения, а затем на его основе генерируете определение OpenAPI.

6.3.1. Генератор OpenAPI с использованием springdoc

Рассмотрим этот сценарий в контексте следующей проблемы: пользователи сообщают, что поля `numberOfReviews` и `reviewRating` возвращаются вашим работающим API, но отсутствуют в документации (ситуация, обратная по отношению к той, которая рассматривалась в сценарии с генерацией). В вашем файле определения OpenAPI используется запись `additionalProperties:false`, из которой следует, что в ответе не должно быть дополнительных свойств, кроме тех, что указаны в определении. А значит, документация не соответствует поведению API. Команда предлагает решить эту проблему генерацией документации API на основе кода приложения.

Чтобы сделать это с использованием Spring Boot, можно подключить библиотеки Java, которые автоматически генерируют определение OpenAPI из проекта, выводя его на основании конфигурации, классов и аннотаций приложения. Два примера таких библиотек для Spring Boot — библиотека Java `springdoc-openapi` (https://springdoc.org) и SpringFox (https://springfox.github.io/springfox). В этом примере я покажу, как это делается в `springdoc-openapi`.

Сгенерированный вариант product-catalog-service уже содержит библиотеку `springdoc-openapi` как зависимость. По умолчанию библиотека предоставляет доступ к определению OpenAPI по адресу http://server:port/context-path/v3/api-docs в формате JSON, а версия в формате YAML доступна по адресу http://server:port/context-path/v3/api-docs.yaml. (Обратите внимание: context-path — имя, используемое для обращения к веб-приложению.) Библиотека также позволяет визуализировать API и взаимодействовать с ним с использованием Swagger UI по адресу http://server:port/context-path/swagger-ui.html. Чтобы включить дополнительную информацию в сгенерированные определения API, можно добавить аннотации Swagger в классы Java (рис. 6.9).

Опробуйте эту возможность на проекте из нашего примера. Запустите приложение из папки `chapter6/product-catalog-service` (если оно не было запущено ранее) командой `mvn spring-boot:run` из командной строки. Перейдите по следующим URL, чтобы просмотреть вывод:

https://liveinternet.club

- http://localhost:8080/v3/api-docs
- http://localhost:8080/v3/api-docs.yaml
- http://localhost:8080/swagger-ui.html

Отлично, вы автоматически сгенерировали свое определение OpenAPI на основе кода!

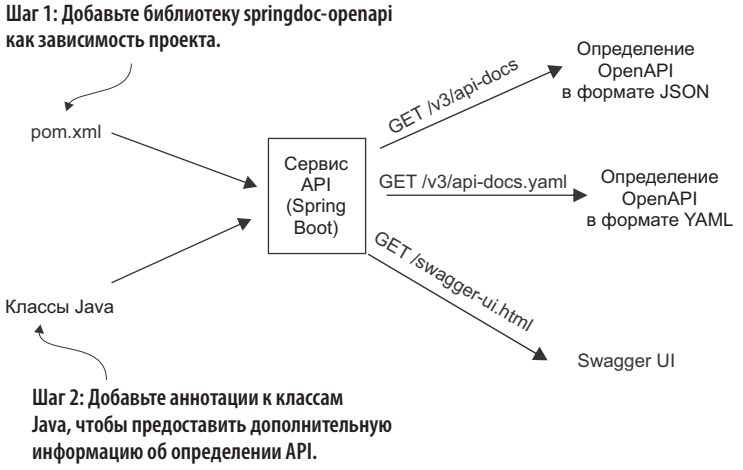


Рис. 6.9. Генерация определений OpenAPI из приложения Spring Boot

6.3.2. Что нужно помнить при генерации определений

При помощи библиотеки `springdoc-openapi` вы добавили информацию с описанием API в код сервиса API Spring Boot и сгенерировали на основе этого кода определение API. Подобные библиотеки существуют и для других статических и динамических языков — например, Django REST Framework (www.django-rest-framework.org/api-guide/schemas) и flask-smorest (https://flask-smorest.readthedocs.io) для приложений Python.

> **СОВЕТ** OpenAPI.tools (https://openapi.tools) — хорошая отправная точка, с которой можно начать поиск информации о других библиотеках генерации OpenAPI и многих других инструментах OpenAPI.

Как и при подходе с генерацией кода, следует обращать внимание на любые ответы API, которые могут поступать от API-шлюза (или другого компонента, находящегося перед сервисом API в стеке). И снова, если API-шлюз существенным образом преобразует сообщения запроса и ответа API, то определение API, сгенерированное на основе сервиса API, не будет точно отражать внешнее поведение API, предоставляемое API-шлюзом.

https://liveinternet.club

Также необходимо добавить в код достаточное описание API, включая примеры значений. По моему опыту, некоторые команды забывают добавить примеры, а также ограничения схемы (`maxLength`, `minLength`, строковые шаблоны и т. д.). Можно работать в итеративном режиме и проверять сгенерированное определение API, чтобы убедиться, что оно содержит информативное описание схемы и примеры, которые будут полезными для читателя документации.

Вместо того чтобы генерировать код на основе определения API, некоторые команды идут другим путем: они пишут код для сервиса API, генерируют на его основе определение OpenAPI, сравнивают сгенерированное определение с тем, которое было получено на этапе проектирования (оно служит источником истины), и устраняют любые отличия. Другие команды сначала создают прототип API в коде, чтобы получить больше информации (технологические ограничения, требования и т. д.), генерируют по прототипу определение API, а затем используют сгенерированное определение для итераций и уточнения API. Хотя этот способ может рассматриваться как схема «сначала код», его также можно считать исследованием в контексте большой неизвестности, позволяющим получить больше информации для поддержки API. Возможность генерации определения из кода в этом сценарии очень полезна, а границы между тем, что традиционно относилось к схемам «сначала код» и «сначала проектирование», при создании определений API постепенно размываются.

Итоги

- Обеспечение соответствия API направлено на согласование опубликованной документации API с поведением сервера API и клиентскими SDK для API.
- Один из способов обеспечить это соответствие основан на генерировании таких артефактов, как код сервиса API, клиентские SDK для API и документация API, на основе определения OpenAPI. При таком подходе определение API считается источником истины для ожидаемого или будущего поведения.
- Преимуществами генерации артефактов на основе OpenAPI являются скорость (по сравнению с написанием тестов схем) и возможность параллельной работы команд с согласованным определением API, которое является источником истины.
- Генерация определения API на основе кода сервиса API (и последующее использование определения для создания документации и клиентских SDK) — другой способ обеспечить соответствие поведения API документации.
- В опубликованной документации API должны учитываться ответы API, поступающие от компонентов за пределами сервиса API (например, API-шлюзов).
- OpenAPI Generator — популярная библиотека для генерации серверного кода, клиентских SDK и документации на основе определений OpenAPI. Библиотека поддерживает различные параметры для настройки сгенерированного кода.

https://liveinternet.club

7 Соответствие API: тестирование схем

В этой главе

- ✓ Написание тестов схем на основе кода
- ✓ Автоматическая генерация тестов схем API
- ✓ Использование валидирующего прокси-сервера для тестов схем

В предыдущей главе вы узнали, как генерация определения OpenAPI на основе сервиса API помогает обеспечить соответствие определения OpenAPI поведению сервиса. Но в некоторых случаях генерация определения OpenAPI на основе сервиса может оказаться невозможной — например, из-за технических ограничений унаследованных сервисов API или организационных ограничений. И даже если вам удастся сгенерировать определение OpenAPI на основе сервиса API, в ходе исследовательского тестирования может обнаружиться неожиданное поведение приложения, не отраженное в определении OpenAPI. Например, из-за запроса с недопустимыми символами сервер приложения на бэкенде может вернуть ошибку перед тем, как запрос будет передан логике обработки ошибок контроллера. Определение OpenAPI, сгенерированное на основе кода контроллера, может не учитывать этот сценарий.

Следовательно, будет полезно создать дополнительный уровень валидации, который будет проверять, что опубликованная документация API соответствует поведению API. В частности, это можно сделать путем тестирования схем (schema testing) с использованием определения OpenAPI как определения схемы API.

https://liveinternet.club

В этой главе мы рассмотрим три метода написания тестов схем OpenAPI: тесты на основе кода, генерируемые тесты и валидирующий прокси-сервер.

Эта глава предназначена для разработчиков, которым нужно создавать тесты схем, но владельцам продуктов API также полезно иметь общее представление о разных способах тестирования схем. Начнем со знакомства с тестированием схем. Затем рассмотрим пример написания тестов схем на основе кода с библиотекой Atlassian Swagger Request Validator. Чтобы продемонстрировать, как генерируются тесты схем на основе файла определения OpenAPI, я приведу примеры с использованием двух инструментов — Schemathesis и Portman. Наконец, чтобы изучить последний метод, я покажу, как подключить валидирующий прокси-сервер Prism к существующему набору сквозных тестов. В завершение главы обсудим, чем тесты схем отличаются от контрактных.

Убедитесь, что вы установили все инструменты из приложения Г. Код этой главы находится в папке `chapter7` проекта примера. Написание тестов требует знания Java. Если у вас нет опыта работы на Java, готовые примеры находятся в папке `chapter7-completed`.

7.1. Тесты схем

Прежде чем обсуждать тестирование схем, разберемся, что же собой представляет схема OpenAPI. Спецификация OpenAPI использует объект схемы для описания типа данных параметров, запросов или ответов в форме, легко читаемой как человеком, так и машиной. Типом данных может быть примитив (целое число, строка и т. д.), массив или объект. Ниже приведен пример объекта схемы в объекте ответа API из файла определения OpenAPI. Это простой объект схемы `Product`, содержащий всего два возможных поля — `id` и `name`:

```
responses:
  "200":
    description: OK                          ← Описание ответа
    content:
      application/json:                      ← Формат ответа
        schema:                              ← Объект схемы ответа
          title: Product
          required:
            - id
          type: object
          additionalProperties: false        ← Ответ содержит только свойства, перечисленные здесь
          properties:
            id:
              type: string
              format: uuid
              description: Идентификатор продукта.
              example: dcd53ddb-8104-4e48-8cc0-5df1088c6113
            name:
              maxLength: 50
              minLength: 1
```

https://liveinternet.club

```
type: string
description: Имя продукта.
example: "Acme Uber Dog Rope Toy"
```

В определении ответного сообщения указано, что успешный ответ 200 возвращает контент типа JSON. Поле `required` схемы показывает, что поле `id` является обязательным, а поле `name` может отсутствовать. Объект схемы задает тип данных полем `type` с возможными значениями `boolean`, `number`, `string`, `array` и `object`. Сообщения, соответствующие этой схеме, могут содержать только свойства `id` и `name`, поскольку схема включает атрибут `additionalProperties: false`, означающий, что сообщения не могут содержать других свойств, кроме определенных. Свойство `id` имеет тип `string` и соответствует формату UUID (Universally Unique IDentifier). Свойство `name` также имеет тип `string` и может содержать от 1 до 50 символов.

В версии 3.0 объекты схемы OpenAPI базируются на спецификации JSON Schema. Так как объекты схемы описывают структуру сообщения, они полезны для автоматизированного тестирования и проверки данных, отправленных клиентом.

Тест схемы для предыдущего сообщения проверяет, что фактическое сообщение, возвращенное операцией API, соответствует схеме. Это автоматизированный тест, который сравнивает ответ API со схемой в определении API и не проходит в случае их несовпадения. Например, для описанной выше схемы тест проверяет, что свойство `name` имеет строковый тип и длину не менее 1 и не более 50. В тесте также можно проверить, что тип контента и заголовки, возвращенные вызовом API, соответствуют заданным в файле определения API.

Определение OpenAPI включает схему для сообщений запросов и ответов. При обсуждении тестов схем в контексте API, документированных в OpenAPI, я использую термин «потребитель» для обозначения клиентского приложения или сервиса, который направляет запрос к API. Термином «провайдер» я обозначаю сервис, предоставляющий ответ API. Этот сервис поддерживает операции API, предоставляющие данные или некоторую функциональность потребителям API. В соответствии с этими определениями схемный тест проверяет, что провайдер следует своему опубликованному контракту провайдера (рис. 7.1).

В отличие от функциональных тестов, тесты схем не проверяют побочные эффекты (например, что объект успешно сохранен в базе данных) или правильность значений в сообщении ответа (например, что значение целочисленного свойства равно 5 или 10). Тесты схем не являются тестами производительности, так что они не выполняют проверки скорости, стабильности или отзывчивости. Тесты схем также следует отличать от контрактных тестов, управляемых потребителем. Контрактное тестирование — более исчерпывающий метод тестирования точек интеграции между двумя программными сервисами, поскольку оно использует конкретные примеры для проверки того, что сервисы потребителя и провайдера одинаково понимают сообщения, которыми они обмениваются в заданных

https://liveinternet.club

сценариях. Тесты схем проверяют только то, что тип данных обмена сообщениями с API соответствует опубликованной документации в некоторый момент времени, а контрактные тесты (с использованием таких инструментов, как Pact и Spring Contracts) идут намного дальше и делают возможным тесное сотрудничество команд и эволюцию взаимодействующих сервисов во времени. Различия между тестами схем и контрактными тестами подробнее обсуждаются ниже.

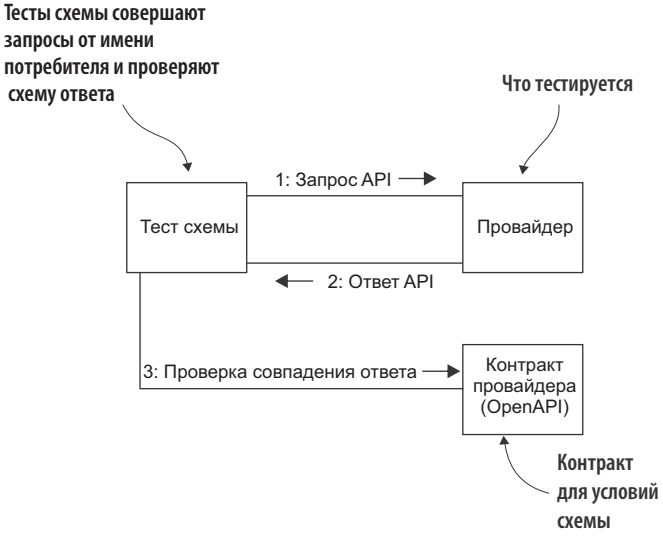


Рис. 7.1. Общая структура теста схемы

> **ПРИМЕЧАНИЕ** Определение OpenAPI может рассматриваться как контракт, и в некоторых источниках в интернете для описания тестов схем используется термин «контрактный тест». Это может создать путаницу, но в книге я различаю эти два термина. Мэтт Феллоуз (Matt Fellows) написал отличный пост по этому поводу: https://mng.bz/z8RQ.

В следующем разделе мы рассмотрим первый метод использования тестов схем для обеспечения соответствия API: тесты схем на основе кода.

7.2. Тесты схем на основе кода

Предположим, пользователи сообщают вам, что поля `numberOfReviews` и `reviewRating` возвращаются вашим API каталога продуктов, но отсутствуют в документации, хотя определение OpenAPI использует запись `additionalProperties:false`, которая означает, что в ответе не должно быть дополнительных свойств, не указанных в определении. Как написать тест схемы для проверки сообщения?

https://liveinternet.club

Автоматизированный, или тест схемы на основе кода, который проверяет тестовые условия для ответа, полученного от сервера API, можно написать двумя способами. Первый — включить отдельные тестовые утверждения в код на основании схемы. Например, можно проверить, что заданное возвращаемое свойство является строкой с максимальной длиной x. В таком случае при изменении схемы в определении OpenAPI придется обновить тест. Второй вариант — включить определение API прямо в тест и использовать библиотеку, которая проверит ответ по определению. Тестовый код может храниться в отдельном тестовом файле с использованием фреймворка юнит-тестирования (например, JUnit в Java) или встраиваться в коллекцию запроса при помощи инструментов клиента API (например, Postman или Insomnia).

Область действия API провайдера — то, что тестируется, — можно изменять. Это может быть полностью развернутый API, включая API-шлюз, так что тест схемы выполняется как сквозной тест, проверяющий весь стек. Также область действия теста схемы может находиться на уровне контроллера; в этом случае проверяется только сервис API с зависимыми мок-системами. Концептуальное представление этой схемы показано на рис. 7.2.

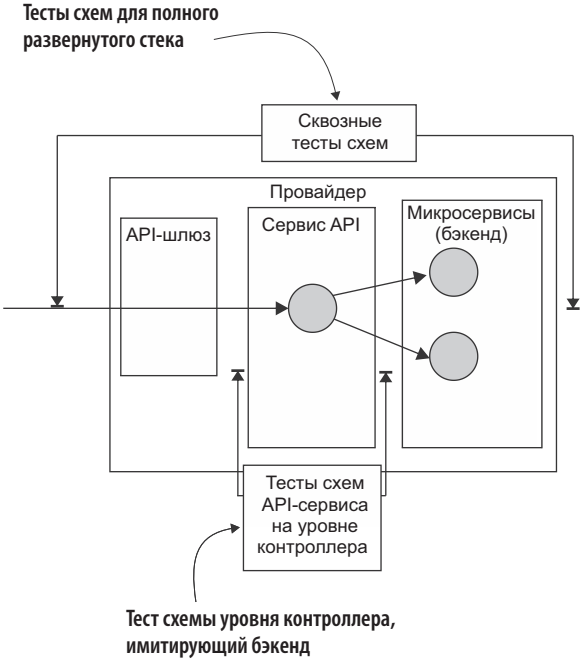


Рис. 7.2. Тест схемы может применяться как ко всему стеку, так и на уровне контроллера сервиса API

https://liveinternet.club

Тест уровня контроллера с мок-зависимостями выполняется быстро, а сквозной тест — медленно, и ему присущи все недостатки сквозных тестов: трудности с отладкой, обслуживанием и масштабированием. Рассмотрим пример написания сквозного теста и теста API уровня контроллера.

7.2.1. Написание сквозных тестов схем

Рассмотрим пример написания теста схемы на основе кода, который работает с эндпоинтом API каталога продуктов и проверяет соответствие ответа файлу определения API `product-catalog-v1-0.oas.yaml`.

Для написания теста Java используем тестовую библиотеку Rest Assured (https://github.com/rest-assured/rest-assured). Rest Assured — предметно-ориентированный язык (DSL, Domain-Specific Language) для тестирования сервисов REST в Java. Он предоставляет динамичный интерфейс для создания тестов API с использованием синтаксиса `given/when/then`. Для проверки ответа по контракту OpenAPI можно воспользоваться открытой библиотекой Atlassian Swagger Request Validator (https://mng.bz/0Gn6).

Для этого теста необходимо запустить Kong. Пользователям систем семейства Unix для запуска API-шлюза Kong следует перейти в каталог `chapter7/kong` и выполнить сценарий `./start_kong.sh`. Также необходимо запустить сервис API — перейдите в каталог `chapter7/product-catalog-api-service` и выполните `./start_api_service.sh`.

> **ПРИМЕЧАНИЕ** Если вы работаете в Windows, сначала задайте свойству `host` в файле `kong.yaml` значение `host.docker.internal`, а затем выполните скрипт `run_kong_docker.bat` из папки `chapter7/kong`, чтобы запустить Kong. В папке `chapter7/product-catalog-api-service` выполните `start_api_service.bat`, чтобы запустить сервис.

Когда API-шлюз и сервис API заработают, можно переходить к написанию сквозного теста схемы. В папке `chapter7/product-catalog-api-service` перейдите в папку `src/test/java` и найдите файл `EndToEndSchemaTest.java`. Используйте Rest Assured DSL, чтобы отправить GET-запрос к эндпоинту `http://localhost:8000/v1/catalog/products/{uuid}` (где `{uuid}` — любое значение UUID), и проверьте, что возвращен код статуса 200 (листинг 7.1). `OpenApiValidationFilter` проверяет, что возвращаемый ответ соответствует файлу определения API `product-catalog-v1-0.oas.yaml`.

Листинг 7.1. Сквозной тест схемы

```
public class EndToEndSchemaTest {
    private final OpenApiValidationFilter validationFilter = new
➥ OpenApiValidationFilter("../product-catalog-v1-0.oas.yaml");
    @Test
    public void testGetProductEndToEnd() {
        given()
                .header("X-API-Key", "my_secret_api_key")
```

Использует DSL-язык Rest Assured

Предоставляет заголовки, необходимые для API-шлюза

https://liveinternet.club

```
                .filter(validationFilter)
                .when()
                .get("http://localhost:8000/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b90fc")
                .then()
                .assertThat()
                .statusCode(200);
    }
}
```

Фильтрация для проверки соответствия определению OpenAPI

Совершает запрос к шлюзу API на порте 8000

Проверяет успешный ответ

Выполните тест, запустив скрипт `./run_end_to_end_schema_test.sh` (или `./run_end_to_end_schema_test.bat` для Windows) из папки `chapter7/product-catalog-api-service`. Тест должен вернуть ошибку следующего вида:

```
[ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed:
➥ 1.58 s <<< FAILURE! - in
➥ com.acmepetsupplies.productcatalog.EndToEndSchemaTest
[ERROR] testGetProductEndToEnd  Time elapsed: 1.577 s  <<< ERROR!
com.atlassian.oai.validator.restassured
➥ .OpenApiValidationFilter$OpenApiValidationException:
{
  "messages" : [ {
    "key" : "validation.response.body.schema.additionalProperties",
    "level" : "ERROR",
    "message" : "Object instance has properties which are not allowed by
➥ the schema[1]: [\"numberOfReviews\",\"reviewRating\"]",
    "context" : {
      "requestPath" : "/v1/catalog/products
➥ /612b4280-b5c0-4ad5-bce7-ede7ab2b90fc",
      "responseStatus" : 200,
      "location" : "RESPONSE",
      "pointers" : {
        "instance" : "/"
      },
      "requestMethod" : "GET"
    }
  } ]
}
    at com.acmepetsupplies.productcatalog.EndToEndSchemaTest
➥ .testGetProductEndToEnd(EndToEndSchemaTest.java:19)
```

Тест показывает, что значение `additionalProperties` (которому присвоено `false`) в определении API конфликтует с API, возвращающим недокументированные поля `numberOfReviews` и `reviewRating`. Обновите тест и задайте в `OpenApiValidationFilter` файл `product-catalog-v1-1.oas.yaml`, в котором определяются два свойства. Снова выполните тест и убедитесь, что он прошел.

Так как тест проверяет ответ, полученный от API-шлюза с точки зрения потребителя API, для него характерны все недостатки сквозных тестов. Они плохо масштабируются с ростом количества тестов, поскольку могут быть медленными;

[1] Object instance has properties which are not allowed by the schema — Экземпляр объекта имеет свойства, которые не разрешены схемой. — *Примеч. пер.*

https://liveinternet.club

их сложно отлаживать в случае провала (из-за большого количества задействованных компонентных сервисов) и для них трудно создавать и обслуживать тестовые данные. Поэтому хорошо подумайте, обоснован ли этот подход.

> **СОВЕТ** В целях безопасности я рекомендую задавать полю `additionalProperties` значение `false`. Это упростит создание тестов, выявляющих, что API возвращает дополнительные данные, о которых вы могли не знать. Некоторые шлюзы позволяют загрузить определение API и использовать его для проверки трафика API на стадии выполнения. Например, это поможет обнаружить аномальное поведение, когда атака заставляет API включать в ответы неожиданные дополнительные данные.

7.2.2. Написание тестов схем уровня контроллера

Аналогичный тест можно написать без тестирования всего стека. Spring Boot позволяет тестировать контроллеры приложений без необходимости запускать сервер для приложения при помощи пакета MockMvc из Spring. Пакет обрабатывает входящие запросы HTTP и передает их контроллеру. В Rest Assured имеется модуль `spring-mock-mvc`, который позволяет провести модульное тестирование контроллеров с использованием MockMvc. Добавьте модуль Rest Assured `spring-mock-mvc` в файл `pom.xml`. Кроме того, добавьте модуль Atlassian `swagger-request-validator-mockmvc` (https://mng.bz/QZ6G), который не только позволяет проверять соответствие запросов и ответов файлам OpenAPI, но и интегрируется со Spring MockMvc:

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>spring-mock-mvc</artifactId>
    <version>5.3.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.atlassian.oai</groupId>
    <artifactId>swagger-request-validator-mockmvc</artifactId>
    <version>${swagger-request-validator.version}</version>
</dependency>
```

Теперь создайте в папке `src/test/java/com/acmesupplies/productcatalog` тест `ControllerSchemaTest`, который проверяет ответ `ProductsApiController`, как показано в листинге 7.2. Тест проверяет ответ при помощи объекта `ResultMatcher`, созданного на основе определения OpenAPI.

Листинг 7.2. Тест схемы контроллера продуктов

```
package com.acmepetsupplies.productcatalog;
import com.acmepetsupplies.productcatalog.controller.ProductsApiController;
import org.junit.Test;
import org.springframework.boot.test.context.SpringBootTest;
import static com.atlassian.oai.validator.mockmvc
➥ .OpenApiValidationMatchers.openApi;
```

https://liveinternet.club

```
import static io.restassured.module.mockmvc.RestAssuredMockMvc.given;
@SpringBootTest
public class ControllerSchemaTest {
    @Test
    public void testGetProductController() {
        given()
                .standaloneSetup(
➥ new ProductsApiController())
                .when()
                .get
➥ ("/v1/catalog/products/612b4280-b5c0-4ad5-bce7-ede7ab2b90fc")
                .then()
                .assertThat()
                .expect(openApi()
➥ .isValid("../product-catalog-v1-0.oas.yaml"))
                .statusCode(200);
    }
}
```

Регистрирует и настраивает контроллер с инфраструктурой Spring MVC

Назначает сопоставитель для проверки соответствия определению OpenAPI

Выполните тест, запустив скрипт `./run_controller_schema_test.sh` (или `./run_controller_schema_test.bat` для Windows) из каталога `chapter7/product-catalog-api-service`. Тест должен вернуть сообщение, похожее на приведенное выше; оно оповещает о двух недостающих полях в определении OpenAPI.

Обратите внимание: тест выполняется достаточно быстро, поскольку это юнит-тест контроллера. В данном примере у `ProductApiController` нет зависимостей, но, если бы они были, имитировать их было бы легко. Так как это быстрый юнит-тест, с его помощью можно тестировать не только тесты «счастливого пути» (happy path tests) — обычно ответы с кодами 2xx, но и тесты «несчастливого пути» (sad path tests) — обычно ответы с кодами 4xx и 5xx. Тем не менее недостаток этого теста заключается в том, что он не проверяет никакое поведение от API-шлюза — например, отправку неверного значения заголовка с ключом API и получение ответа об ошибке 401 аутентификации ключа API.

7.3. Что необходимо учитывать при использовании тестов схем на основе кода

Вы увидели, что тесты схем на основе кода могут быть в форме сквозных тестов или тестов уровня контроллера. Следует помнить о затратах на обслуживание этих тестов. Каждый раз, когда вы добавляете новый эндпоинт в API, необходимо добавить и тесты ответных сообщений для разных позитивных и негативных сценариев для этого эндпоинта, что может означать увеличение объема кода тестов, который придется обслуживать.

Такой подход может пригодиться, если надо объединить несколько вызовов API в один тестовый сценарий, который проверяет более глубокие части логики приложения, необходимые для возврата объектов в сложном состоянии. Создайте последовательность вызовов API на программном уровне и протестируйте ответ.

https://liveinternet.club

В рассмотренном нами примере на Java для написания тестов схем использовалась библиотека Atlassian Swagger Request Validator. Среди других открытых библиотек, предоставляющих аналогичные возможности, можно выделить библиотеку Java Hikaku (https://github.com/codecentric/hikaku) и библиотеку JavaScript jest-openapi (https://mng.bz/X1yv). Как уже говорилось, можно писать сквозные тесты API на JavaScript на таких платформах API, как Postman и Insomnia. Пример проверки ответов в Postman см. на странице https://mng.bz/y8RB.

7.4. Генерация тестов схем

А если вместо того, чтобы писать тесты схем для каждого эндпоинта и каждого типа ответа эндпоинта, генерировать тесты из определения OpenAPI? Для этого нужен инструмент, который генерирует и выполняет тесты схем для счастливого и несчастливого пути на основе определения API, проверяя API-шлюз или API-сервис. На рис. 7.3 показано, как работает эта схема. Сгенерированные тесты можно сохранить на будущее в системе контроля версий или любом другом месте.

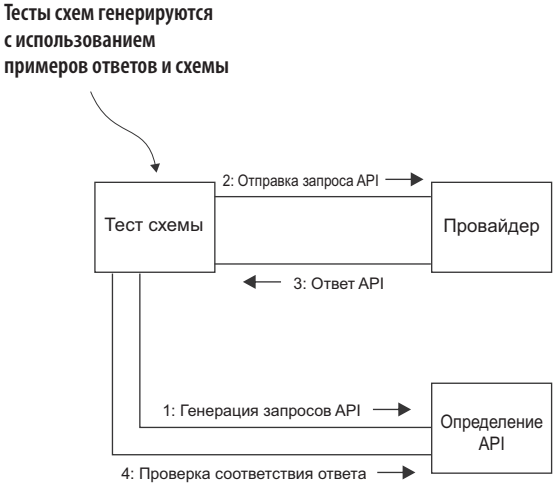


Рис. 7.3. Генерация тестов схем на основе определения OpenAPI

Этот подход используется в таких инструментах, как Schemathesis и Portman. Начнем с Schemathesis.

7.4.1. Использование Schemathesis для тестирования схем

Schemathesis (https://schemathesis.readthedocs.io) — инструмент тестирования API, который генерирует и запускает тесты на основании предоставленного файла определения API. Он используется для фаззинг-тестирования и при

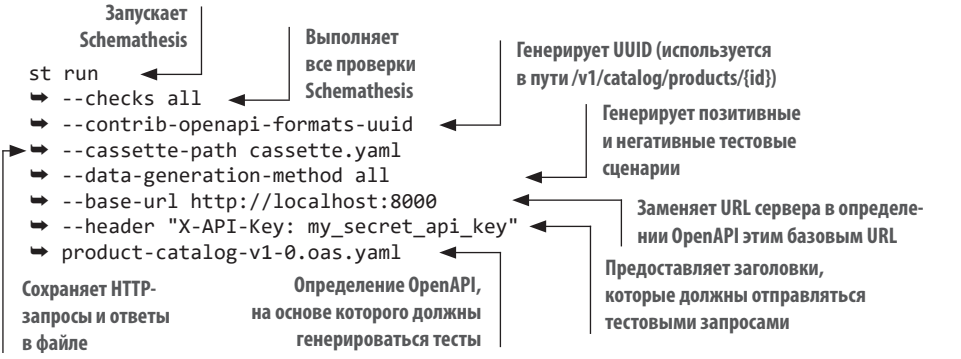
https://liveinternet.club

этом также проверяет схему API. Schemathesis написан на Python и строится на основе библиотеки тестирования Hypothesis, но вы можете воспользоваться им для тестирования любых API, определенных в документе OpenAPI, и для тестирования GraphQL API.

Допустим, вас просят написать тест схемы для проверки API каталога продуктов на основе файла определения API `product-catalog-v1-0.oas.yaml`. Чтобы сделать это с помощью Schemathesis, необходимо запустить Schemathesis CLI и передать ему файл определения API.

Запустите Schemathesis CLI и предоставьте эту стратегию как скрипт, который должен выполняться перед выполнением тестов, как показано в листинге 7.3. (Помните, что при этом должен работать API-шлюз; чтобы запустить его, перейдите в каталог `chapter7/kong` и выполните скрипт `./start_kong.sh`. Также должен работать сервис API; чтобы запустить его, перейдите в каталог `chapter7/product-catalog-api-service` и выполните скрипт `./start_api_service.sh`.) Скрипт указывает, что CLI должен выполнить все проверки Schemathesis и записать тестовые запросы в файл `cassette.yaml`.

Листинг 7.3. Выполнение Schemathesis



СОВЕТ Также можно воспользоваться другим способом: выполнить вспомогательный скрипт `./run_schemathesis_for_v1_0_api.sh` в корне папки `chapter7`.

Тест должен провалиться со следующим сообщением:

```
- Response violates schema
    Additional properties are not allowed[1]
➥ ('numberOfReviews', 'reviewRating' were unexpected)
```

Schemathesis помог найти проблему со схемой! Он использовал примеры, представленные в файле определения OpenAPI, при отправке запросов API.

[1] Response violates schema. Additional properties are not allowed — Ответ нарушает схему. Дополнительные свойства недопустимы. — *Примеч. пер.*

https://liveinternet.club

Просмотрите файл `product-catalog-v1-0.oas.yaml`. Вы увидите, что UUID, заданный для `productId`, а именно `c05aed25-97cd-4cbc-b299-3796538eee9c`, используется для отправки запроса в файле `cassette.yaml`. Следовательно, полезно включать примеры значений в файл определений API — как для пользователей, которые будут читать документацию API, так и для автоматизированных инструментов (таких, как Schemathesis).

> **СОВЕТ** Если вы хотите, чтобы Schemathesis игнорировал примеры, явно заданные в документе OpenAPI, и сгенерировал новые значения, передайте в командной строке аргумент `--hypothesis-no-phases explicit`.

Теперь вместо `product-catalog-v1-0.oas.yaml` выполните тест для файла `product-catalog-v1-1.oas.yaml`, содержащий исправления схемы. Для этого можно запустить скрипт `chapter7/run_schemathesis_for_v1_1_api.sh`. Результат будет выглядеть примерно так:

```
==============SUMMARY =================
Performed checks:
    not_a_server_error             101 / 101 passed          PASSED
    status_code_conformance        101 / 101 passed          PASSED
    content_type_conformance       101 / 101 passed          PASSED
    response_headers_conformance   101 / 101 passed          PASSED
    response_schema_conformance    101 / 101 passed          PASSED
Network log: cassette.yaml
```

Что он означает? Schemathesis сгенерировал и выполнил тесты для API (в данном случае 101, но в вашем запуске число может быть другим). В каждом из тестов были выполнены пять видов проверок.

- Если сервер отвечает кодом HTTP-статуса 5xx, Schemathesis помечает тест как проваленный с ошибкой `not_a_server_error`.
- Если сервер отвечает кодом статуса, не заданным в файле определения API, тест помечается как проваленный с ошибкой `status_code_conformance`.
- Если сервер отвечает типом контента, не заданным в файле определения API, тест помечается как проваленный с ошибкой `content_type_conformance`.
- Если ответ сервера содержит заголовок, не заданный в файле определения API, тест помечается как проваленный с ошибкой `response_headers_conformance`.
- Если тело ответа сервера не соответствует схеме, заданной в файле определения API, тест помечается как проваленный с ошибкой `response_schema_conformance`.

Просмотрите в файле `chapter7/cassette.yaml` каждый тестовый запрос, выполненный Schemathesis, и ответ, полученный от сервера. Файл `cassette.yaml` имеет формат VCR (https://github.com/vcr/vcr). Он содержит массив элементов `http_interactions`. Каждый элемент `http_interaction` содержит поля с уникальным идентификатором, временем выполнения теста, статусом проверок

https://liveinternet.club

Schemathesis для этого теста, а также подробности запроса и ответа. В своем запуске Schemathesis для этого примера я получил следующее содержимое файла `cassette.yaml`:

```
{"id": "791e653b-32c4-4602-a2db-99a95cb5e78f", "status": 400,
➥ "code":"validation.client_error", "title": "Bad Request",
➥ "detail": "Illegal character CNTL=0x1b"}
```

Обратите внимание: для этой ошибки поле `detail` содержит причину сбоя (отправка недопустимого символа в запросе). Будучи инструментом фаззинг-тестирования, Schemathesis отправляет запрос с недопустимым символом. Инструменты фаззинг-тестирования API ищут дефекты в API, генерируя и отправляя случайные запросы API, и некоторые из запросов включают некорректно оформленные или частично оформленные данные. Обычно для таких ошибок Jetty (сервер приложения в нашем примере Spring Boot) возвращает ошибку страницы HTML 400, что не соответствует ответу в определении OpenAPI. Но я задал в классе `JettyCustomizer.java` обработчик `CustomErrorHandler`, который переопределяет это поведение и возвращает сообщение `application/json`, соответствующее файлу определения API.

Инструменты фаззинг-тестирования — такие, как Schemathesis, — отлично подходят для нахождения подобных граничных случаев, которые легко упустить в других ситуациях.

> **СОВЕТ** В Schemathesis можно настроить запрет генерации недопустимых символов; для этого используется флаг командной строки `--generation-allow-x00 false`.

Schemathesis помогает убедиться, что API эффективно проверяет ввод, поскольку он отправляет независимые запросы API со случайными входными данными и проверяет, что ответ соответствует определению API. Тем не менее в некоторых случаях для того, чтобы отправить успешный запрос API и протестировать схему ответа, приложение должно находиться в определенном состоянии, обусловленном предыдущими вызовами API. Иначе говоря, вызовы API следует тестировать в комбинациях. Например, чтобы получить профиль пользователя, необходимо сначала отправить вызов API для создания пользователя, а затем передать возвращенный идентификатор пользователя в следующем запросе API на получение профиля. Чтобы выполнять подобные тесты с состоянием в Schemathesis, следует передавать отношение между операциями, определяя объекты связей в документе OpenAPI 3.0. Связь OpenAPI представляет определенные отношения и структуру перемещения между ответами и другими операциями API. (Обсуждение связей выходит за рамки книги, но вы можете больше узнать о них в документации Swagger: https://swagger.io/docs/specification/links/.) Когда в документе OpenAPI присутствуют связи, Schemathesis использует данные в ответе API для отправки последующих вызовов API вместо генерации случайных данных.

Schemathesis написан на Python, и вы можете написать тесты Python, использующие интерфейс Schemathesis, вместо CLI. Библиотека Schemathesis

https://liveinternet.club

имеет программный API, который позволяет выполнять все те же операции, что и с CLI.

В этом разделе я использовал Schemathesis, чтобы показать, как генерировать и запускать тесты схемы на основе определений OpenAPI.

Есть еще один интересный инструмент, использующий тот же подход, — Portman. Рассмотрим его в следующем разделе.

7.4.2. Использование Portman для тестирования схем

Portman (https://github.com/apideck-libraries/portman) — инструмент с интерфейсом командной строки (CLI), который генерирует и выполняет коллекции Postman для проверки того, что определение OpenAPI соответствует реализации API. Portman позволяет импортировать сгенерированную коллекцию в десктопный клиент Postman или отправить ее в рабочее пространство Postman. Также коллекцию можно выполнить. Это позволяет включить Portman в пайплайн CI/CD.

Используем Portman для тестирования счастливого пути (успешный ответ с кодом 200) по файлу `product-catalog-v1-1.oas.yaml`; для этого выполните следующую команду из каталога `chapter7`:

```
portman --cliOptionsFile portman/portman-cli.json
```

Отчет о запуске теста будет включать фрагмент следующего вида:

```
Product Catalog API
❑ Products
↳ View a product's details
  GET http://localhost:8000/v1/catalog/
➥ products/c05aed25-97cd-4cbc-b299-3796538eee9c [200 OK, 443B, 46ms]
  ✓  [GET]::/v1/catalog/products/:id - Status code is 2xx
  ✓  [GET]::/v1/catalog/products/:id - Schema is valid
  ✓  [GET]::/v1/catalog/products/:id - Content-Type is application/json
  ✓  [GET]::/v1/catalog/products/:id - Response has JSON Body
Collection run completed.
```

Portman использует первое значение-пример в определении эндпоинта для запуска теста. Файл `Portman-cli.json` содержит параметры командной строки Portman. В данном примере он предоставляет Portman местоположение файла определения OpenAPI для тестирования, а также имя, которое должно быть присвоено созданному файлу Postman. Также предоставляется ссылка на файл среды, в котором хранятся любые переменные Postman для сгенерированной коллекции Postman. Кроме того, передаются другие параметры CLI: `runNewman`, чтобы Portman выполнил тесты с использованием встроенного исполнителя Newman (https://github.com/postmanlabs/newman) для выполнения коллекций Postman в командной строке, `includeTests` для включения тестов согласованности Portman в сгенерированную коллекцию Postman и `baseUrl` для переопределения URL сервера при выполнении коллекции Postman. Файл `Portman-cli.json` выглядит так:

https://liveinternet.club

```
{
  "local": "product-catalog-v1-1.oas.yaml",
  "output": "collection.postman.json",
  "envFile": "portman/.env-portman",
  "portmanConfigFile": "portman/portman-config.json",
  "includeTests": true,
  "syncPostman": false,
  "runNewman": true,
  "baseUrl": "http://localhost:8000"
}
```

Поле `portmanConfigFile` содержит ссылку на файл конфигурации Portman, в котором хранится информация о выполняемых тестах соответствия. Portman использует термин «контрактные тесты» для описания тестов, проверяющих, что API соответствует определению OpenAPI. В нашем примере файл конфигурации `portman/portman-config.json` указывает, что для всех операций API, заданных в определении API, Portman должен проверить, что код статуса, схема, тип контента и заголовки заданы так же, как в определении. Также он должен проверить, что ответ API имеет формат JSON. Вот как это может выглядеть:

```
{
  "version": 1.0,
  "overwrites": [
    {
      "openApiOperation": "*::/*",
      "overwriteRequestHeaders": [
        {
          "key": "Host",
          "value": "api.acme-pet-supplies.com",
          "overwrite": true
        }
      ]
    }
  ],
  "tests": {
    "contractTests": [
      {
        "openApiOperation": "*::/*",
        "statusSuccess": {
          "enabled": true
        }
      },
      {
        "openApiOperation": "*::/*",
        "schemaValidation": {
          "enabled": true
        }
      },
      {
        "openApiOperation": "*::/*",
        "contentType": {
          "enabled": true
        }
      },
```

Задает пользовательские изменения, вносимые в данные запроса в сгенерированной коллекции Postman

Применяет следующую перезапись ко всем операциям API

Перезаписывает заголовок запроса

Перезаписывает значение заголовка Host

Проверяет, что статус ответа соответствует документации

Проверяет, что схема ответа соответствует документации

Проверяет, что возвращенный тип контента соответствует документации

https://liveinternet.club

```
      {
        "openApiOperation": "*::/*",
        "jsonBody": {
          "enabled": true
        }
      },
      {
        "openApiOperation": "*::/*",
        "headersPresent": {
          "enabled": true
        }
      }
    ]
  },
  "globals": {
    "securityOverwrites": {
      "apiKey": {
        "key": "X-API-Key",
        "value": "{{apiKey}}"
      }
    }
  }
}
```

Проверяет, что возвращенный контент имеет формат JSON

Проверяет, что заголовки присутствуют

Задает имя и значение заголовка ключа API

Задает заголовку ключа API значение переменной среды PORTMAN_API_KEY, определенной в .env-portman

СОВЕТ В Portman CLI предусмотрена команда инициализации для запуска с созданием конфигурации. Опробуйте эту возможность. Выполните команду `portman --init` и действия по воссозданию конфигурации.

Возможности Portman по генерированию тестов схем как коллекций Postman удобны для разработчиков, использующих коллекции. Portman можно настроить для синхронизации сгенерированных коллекций с рабочим пространством Postman, чтобы сделать возможным совместное использование коллекций.

7.5. Использование валидирующего прокси-сервера для проверки соответствия

Еще один подход к тестированию соответствия основан на применении паттерна «Валидирующий прокси-сервер». В этом разделе будут рассмотрены сам паттерн и конкретный пример его использования.

Один из способов проверить, что схема соответствует поведению API, основан на передаче запросов и ответов API через промежуточный сервер, который проверяет соответствие ответа от сервера API ожидаемому определению API. Промежуточный (прокси-) сервер может сообщить об ошибке, если ответ сервера API не соответствует ожидаемой схеме. Для этого прокси-серверу необходима ссылка на определение OpenAPI, содержащее схему ответа. Структура паттерна изображена на рис. 7.4.

Одно из преимуществ такого подхода в том, что он позволяет легко начать работу, подключив валидирующий прокси-сервер к существующему семейству тестов API, особенно сквозных. Однако покрытие тестирования схем будет

https://liveinternet.club

соответствовать покрытию сквозных тестов и может оказаться неполным — например, не включать все возможные негативные сценарии для каждого эндпоинта. И конечно, необходимо учитывать все сопутствующие недостатки (хрупкость, сложности с отладкой, затрудненное масштабирование и т. д.).

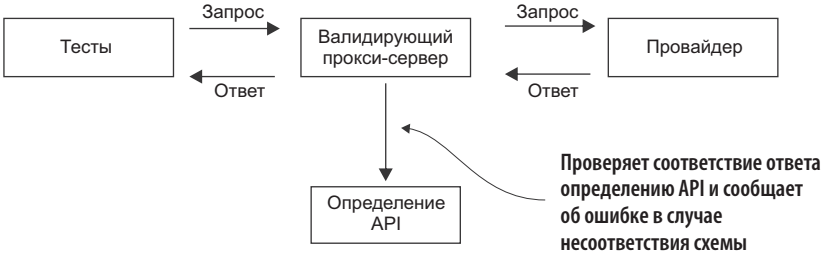


Рис. 7.4. Использование валидирующего прокси-сервера для тестирования соответствия API

Рассмотрим пример этого подхода в действии. Возьмем предыдущий сценарий, в котором поля `numberOfReviews` и `reviewRating` были документированы в файле определения API каталога продуктов. Как тест, использующий валидирующий прокси-сервер, может обнаружить это? Начнем с создания сквозного теста `ProductsApiEndToEndTest`, который отправляет запрос к API-шлюзу и проверяет успешный ответ 200 OK, как показано в листинге 7.4.

Листинг 7.4. Сквозной тест на языке Java

```
package com.acmepetsupplies.productcatalog;
import org.junit.Test;
import static io.restassured.RestAssured.given;
public class ProductsApiEndToEndTest {
    @Test
    public void testGetProductEndToEnd() {
        given()
                .header("X-API-Key", "my_secret_api_key")
                .when()
                .baseUri("http://127.0.0.1:8000")
                .get("/v1/catalog
➥ /products/612b4280-b5c0-4ad5-bce7-ede7ab2b90fc")
                .then()
                .log()
                .body()
                .assertThat()
                .statusCode(200);
    }
}
```

Базовый URI указывает на API-шлюз

Записывает в журнал тело ответа

Запустите тест и убедитесь, что возвращается успешный ответ, — перейдите в `chapter7/product-catalog-api-service` и выполните скрипт `./run_products_api_end_to_end_test.sh` (или `./run_products_api_end_to_end_test.bat` для

https://liveinternet.club

Windows). В выходных данных обратите внимание на то, что тело ответа сохранено в журнале.

Затем необходимо изменить эндпоинт сквозного теста, чтобы вместо него использовалась ссылка на валидирующий прокси-сервер. Prism (https://github.com/stoplightio/prism) — валидирующий прокси-сервер и мок-сервер для определений OpenAPI и коллекций Postman. Если вы выполнили инструкции из приложения Г, Prism уже установлен на вашем компьютере. Перейдите в папку `chapter7`, запустите Prism и передайте ему ссылки на определение OpenAPI и API-шлюз. Кроме того, установите флаг `--errors`, чтобы при нарушениях в ответе выводилось сообщение об ошибке:

```
prism proxy
➥ product-catalog-v1-0.oas.yaml
➥ http://127.0.0.1:8000
➥ --errors
```

По умолчанию Prism запускается на порте 4010. Теперь, когда у вас имеется работающий сервер Prism, измените значение `baseUri` в классе `ProductsApiEndToEndTest` с `http://127.0.0.1:8000` на `http://127.0.0.1:4010`, чтобы тестовый запрос передавался Prism, а не API-шлюзу. Снова запустите тест `./run_products_api_end_to_end_test.sh`; вы увидите, что он не проходит. Для тела ответа регистрируется следующее сообщение:

```
{
    "type": "https://stoplight.io/prism/errors#VIOLATIONS",
    "title": "Request/Response not valid[1]",
    "status": 500,
    "detail": "Your request/response is not valid and the
➥ --errors flag is set, so Prism is generating this error for you.[2]",
    "validation": [
        {
            "location": [
                "response",
                "body"
            ],
            "severity": "Error",
            "code": "additionalProperties",
            "message": "must NOT have additional properties;
➥ found 'reviewRating'[3]"
        },
        {
            "location": [
                "response",
                "body"
            ],
```

[1] Request/Response not valid — Запрос/ответ недействителен. — *Примеч. пер.*

[2] Your request/response is not valid and the errors flag is set, so Prism is generating this error for you — Prism сгенерировал ошибку, поскольку ваш запрос/ответ недействителен и установлен флаг ошибок. — *Примеч. пер.*

[3] Must NOT have additional properties; found 'reviewRating' — Не должно содержать дополнительных свойств; найдено значение 'reviewRating'. — *Примеч. пер.*

https://liveinternet.club

```
            "severity": "Error",
            "code": "additionalProperties",
            "message": "must NOT have additional properties;
➥ found 'numberOfReviews'[1]"
        }
    ]
}
java.lang.AssertionError: 1 expectation failed[2].
Expected status code <200> but was <500>[3].
```

Сообщения об ошибках в Prism выводятся в формате RFC7807 Problem Details и содержат свойства `type` (URI, идентифицирующий тип проблемы), `title` (краткое описание типа проблемы), `status` (код статуса HTTP) и `detail` (объяснение проблемы). Prism возвращает ошибку 500 типа VIOLATION, потому что ответ от API-шлюза не соответствует определению API, а при запуске Prism был указан флаг `--errors`. Свойства `validation.location` и `validation.message` предоставляют детали места в определении API, инициировавшего ошибку, и подробное сообщение о причинах возникновения ошибки.

Кроме того, обратите внимание, что сообщение об ошибке в Prism содержит подробную информацию о взаимодействиях запроса и ответа с сервером API-шлюза:

```
[07:42:50] › [HTTP SERVER] get /v1/catalog/products/9764d95c-7757-4a4d-
➥ a800-660b8fe9392b ?  info      Request received[4]
[07:42:50] ›     [PROXY] ?  info      Forwarding "get" request to[5]
➥ http://localhost:8000/v1/catalog/products/9764d95c-7757-4a4d-a800-
➥ 660b8fe9392b...
[07:42:50] ›     [PROXY] ?  info      The upstream call to
➥ /v1/catalog/products/9764d95c-7757-4a4d-a800-660b8fe9392b
➥ has returned 200[6]
[07:42:50] › [HTTP SERVER] get /v1/catalog/products/9764d95c-7757-4a4d-
➥ a800-660b8fe9392b ?  error     Request terminated with error[7]:
➥ https://stoplight.io/prism/errors#VIOLATIONS:
➥ Request/Response not valid
```

Вы можете протестировать, что произойдет при выполнении Prism с файлом определения API, соответствующим вышестоящему серверу. Остановите работающий экземпляр Prism и перезапустите его с файлом `product-catalog-v1-1.oas.yaml`, как показано ниже:

---

[1] Must NOT have additional properties; found 'numberOfReviews' — Не должно содержать дополнительных свойств; найдено значение 'numberOfReviews'. — *Примеч. пер.*

[2] 1 expectation failed — Нарушено одно предположение. — *Примеч. пер.*

[3] Expected status code <200> but was <500> — Вместо ожидаемого кода <200> получен код <500>. — *Примеч. пер.*

[4] Request received — Запрос получен. — *Примеч. пер.*

[5] Forwarding “get” request to — Направление запроса “get” к. — *Примеч. пер.*

[6] The upstream call to /v1/catalog/products/9764d95c-7757-4a4d-a800-660b8fe9392b has returned 200 — Восходящий вызов /v1/catalog/products/9764d95c-7757-4a4d-a800-660b8fe9392b возвращает ответ 200. — *Примеч. пер.*

[7] Request terminated with error — Запрос завершился с ошибкой. — *Примеч. пер.*

https://liveinternet.club

```
prism proxy
➥ product-catalog-v1-1.oas.yaml
➥ http://127.0.0.1:8000
➥ --errors
```

Теперь снова выполните тест `./run_products_api_end_to_end_test.sh`, и вы увидите, что тест проходит успешно.

7.6. Сравнение методов соответствия API

В этой главе были рассмотрены различные методы написания тестов схем. В табл. 7.1 собраны преимущества и недостатки этих методов.

Таблица 7.1. Сравнение методов тестирования схем API

	Сквозные тесты схем на основе кода	Тесты схем уровня контроллера на основе кода	Автоматически генерируемые сквозные тесты схем с фаззинг-тестированием	Сквозные тесты схем с валидирующим прокси-сервером
Интегрируется в существующее семейство сквозных тестов без написания нового кода	—	—	—	Да
Гарантирует соответствие схемы ответам по всему стеку	Да	Нет	Да	Да
Фаззинг-тестирование может выявлять непредвиденные ответы	Нет	Нет	Да	Если выполняется как часть фаззинг-тестирования
Быстрая обратная связь благодаря тестированию сервиса в изоляции	Нет	Да	Нет	Нет
Не требует изменения кода для новых ответов или эндпоинтов	Нет	Нет	Да. Тесты могут генерироваться по документации API, но могут требовать специальной настройки конфигурации	Prism не требует изменений, но сами сквозные тесты требуют изменения для тестирования новых эндпоинтов или ответов

Как тестирование схем, так и генерация определения API на основе кода (рассматриваемая в главе 6) ориентированы на решение проблемы соответствия API. В табл. 7.2 собраны преимущества и недостатки этих методов. На практике для обеспечения хорошего соответствия API желательно объединять оба подхода.

https://liveinternet.club

Таблица 7.2. Сравнение методов генерации и тестирования схем для проверки соответствия API

	Генерация определения OpenAPI на основе кода	**Тестирование схем определения OpenAPI**
Скорость выполнения	Генерируется быстро в процессе сборки	Может быть быстрым при выполнении на уровне контроллера, но более медленным при выполнении как части сквозного теста
Полнота	Покрывает схему только контроллеров сервисов API, по которым генерируется, и не покрывает поведение API-шлюзов или других промежуточных компонентов (таких, как брандмауэры или промежуточные прокси-серверы)	Делает исчерпывающие предположения по ответам, получаемым потребителем, что обеспечивает покрытие всего стека
Затраты на обслуживание	Низкие. При выполнении как части пайплайна CI/CD может заново генерировать схему как этап пайплайна каждый раз, когда вносится изменение	При выполнении в составе сквозного теста затраты на обслуживание будут привязаны к затратам на обслуживание сквозного теста, которые могут быть высокими
Простота освоения на начальном этапе	Там, где один концептуальный API обслуживается несколькими сервисами API, генерация определения OpenAPI для API потребует объединения сгенерированных определений для нескольких сервисов. Генерация схемы может требовать расстановки аннотаций в контроллере сервиса API и в коде модели ответа	Может запускаться достаточно быстро — например, при использовании валидирующего прокси-сервера с существующими сквозными тестами, и может иметь специализированное семейство тестов, которое выполняет тестирование схем по всему API

7.7. Отличие тестов схем от контрактных тестов, управляемых потребителем

В начале этой главы я упомянул о том, что тестирование схем отличается от контрактного тестирования, управляемого потребителем. Прежде чем объяснять, чем отличается тестирование схем, сначала стоит разобраться, что же собой представляет контрактное тестирование.

Представьте систему, в которой несколько эволюционирующих сервисов взаимодействуют друг с другом. Как проверить, что новая версия сервиса все еще может взаимодействовать с другими сервисами? Один из возможных вариантов — развернуть все сервисы и выполнить интеграционный тест. Но с ростом числа взаимодействующих сервисов интеграционные тесты усложняются. Чтобы выполнить интеграционный тест, придется сначала развернуть и настроить все сервисы в среде, подобной продакшен. Сами интеграционные тесты могут

https://liveinternet.club

выполняться достаточно долго, становятся хрупкими, создают сложности с обслуживанием и масштабированием.

Контрактное тестирование, управляемое потребителем, представляет иной подход к интеграционному тестированию. Оно подразумевает изолированное тестирование сервиса относительно потребительского контракта — документа, определяющего обмен сообщениями, которые должны поддерживаться провайдером. Иначе говоря, это способ убедиться, что два сервиса одинаково понимают, как они должны взаимодействовать друг с другом. Для этого каждый из них тестируется в изоляции с использованием конкретных сообщений (примеров из спецификации). Документ, содержащий эти сообщения, служит контрактом, определяющим предполагаемые взаимодействия сообщений (рис. 7.5).

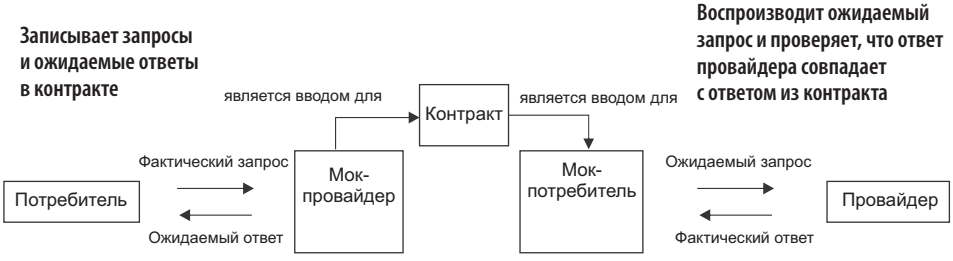


Рис. 7.5. Контрактное тестирование, управляемое потребителем

В контрактных тестах, управляемых потребителем, потребитель передает свои ожидания о поведении API провайдера с контрактом потребителя. Контракт потребителя документирует обмены сообщениями — последовательность или историю передаваемых сообщений, необходимых для перевода сервиса-провайдера в определенное состояние. В обмене сообщениями тестируются только возможности провайдера, используемые потребителями. Это означает, что неиспользуемая функциональность провайдера может эволюционировать без нарушения прохождения тестов. Потребительские контракты стимулируют обсуждение между командами разработчиков и их сотрудничество в отношении эволюции сервисов. Два примера инструментов контрактного тестирования — Pact (https://pact.io) и Spring Cloud Contracts (https://spring.io/projects/spring-cloud-contract). Контракты потребителя, описывающие обмены сообщениями (для запроса A вы получаете ответ B), отличаются от файлов определения API, которые ограничиваются описанием структуры и формата API. Тестирование файлов определения API проверяет, что поведение провайдера соответствует его опубликованной схеме.

Контрактное тестирование, управляемое потребителем, хорошо подходит для тестирования интеграций между сервисами в организации. Оно может заменить многие сквозные тесты, проверяющие интеграции (например, тестирование интеграций

https://liveinternet.club

микросервисов). В таком сценарии команда провайдера может управлять отношениями с небольшим набором известных приложений-потребителей API.

> **СОВЕТ** За дополнительной информацией о контрактном тестировании, управляемом потребителем, обращайтесь к книге Марка Винтерингема (Mark Winteringham) «Testing Web APIs»[1] (Manning, 2022; www.manning.com/books/testing-web-apis).

Тестирование схем определений API для проверки соответствия более уместно для внешних API, где использование контрактов потребителей непрактично. У внешних API может быть много потребляющих сервисов, и для команд потребителей будет непрактично управлять своими индивидуальными требованиями через контракты потребителей — в конце концов, команды потребителей могут даже не использовать тот же инструмент контрактного тестирования, управляемого потребителем, что и команда провайдера. При отсутствии контрактов потребителей для внешних API важно опубликовать точный файл определения API, описывающий интерфейс провайдера, его возможности и схему сообщений. Тестирование схем провайдера поможет убедиться в точности определения API.

Итоги

- Тесты схем предназначены для проверки соответствия API их определению в сценариях, когда не удается применить контрактные тесты, управляемые потребителем, — например, для внешних или общедоступных API.
- Тесты схем на основе кода подразумевают написание тестового кода, который проверяет предположения, сравнивающие тип данных сообщения, возвращаемого провайдером, со схемой из своего опубликованного определения API.
- Тесты схем на основе кода могут выполняться либо как сквозные, либо на уровне контроллера сервиса API. Тесты уровня контроллера выполняются быстрее и не имеют многих недостатков сквозных тестов.
- Генерация тестов схем по определению OpenAPI избавляет разработчиков от необходимости писать код при каждом изменении API. Такие инструменты, как Schemathesis и Portman, могут генерировать и выполнять позитивные и негативные сценарии на основании определения API и проверять их для сервера API.
- Валидирующий прокси-сервер может находиться между автоматизированным семейством тестов и сервером API. Он проверяет, что ответ от сервера соответствует схеме. Этот способ позволяет быстро интегрировать тестирование схем в рабочий поток, поскольку прокси-сервер может быть включен в существующее семейство сквозных тестов.
- Сочетание генерации определений OpenAPI с тестированием схем помогает обеспечить соответствие для внешних API.

[1] Винтерингем М. «Тестирование веб-API». СПб., издательство «Питер».

https://liveinternet.club

8 CI/CD для артефактов API. 1: управление на этапе исходного кода

В этой главе

- ✓ Определение автоматизированных средств управления для исходных файлов API-артефактов в пайплайне
- ✓ Выполнение средств управления в пайплайне
- ✓ Создание мультифайловых определений API и операции с ними

В больших программных проектах могут быть задействованы несколько команд, создающих для организации много внешних RESTful API. Централизованное ручное управление — один из способов убедиться, что API удовлетворяют стандартам соответствия и удобства использования, принятым в организации. Но, как обсуждалось в главе 1, централизованное ручное управление не масштабируется, а это приводит к снижению гибкости, увеличивает время загрузки и раздражает разработчиков. Как команды управления могут сократить необходимость в ручных дизайн-ревью API и внедрить последовательные, автоматизированные самообслуживаемые ревью для поддержки различных команд разработки?

Автоматизированное управление — не единственная проблема. В процессе создания программного продукта на основе файлов определений API генерируются различные *артефакты API*. К ним относятся файлы определений API и побочные продукты, генерируемые на их основе и предоставляемые потребителям API и другим внутренним стейкхолдерам. В частности,

https://liveinternet.club

документация API, клиентские SDK, альтернативные форматы определений API, отчеты и т. д. Если команды разработчиков создают эти артефакты с использованием ручных и ситуативных процессов, это вредит единообразию. Как же обеспечить последовательный механизм генерации артефактов API для всех команд в организации?

В этой главе мы обсудим, как внедрить управление артефактами API в существующий *пайплайн развертывания* (deployment pipeline). Пайплайн развертывания — также называемый *пайплайном непрерывной интеграции и непрерывной доставки* (CI/CD, continuous шntegration and continuous delivery) или *пайплайном доставки* (delivery pipeline) — представляет собой набор процессов, которые переносят программные изменения из коммитов в системе контроля версий в выпускаемый программный продукт надежным и воспроизводимым образом. Мы обсудим, как пайплайн развертывания обеспечивает централизованное, автоматизированное и быстрое предоставление проектировщикам API обратной связи о целостности и соответствии API, а также других метрик оценки определений API. Также я покажу, как использовать пайплайн для генерации и сохранения артефактов API, готовых к публикации.

Для чего нужно управлять артефактами API в пайплайне CI/CD? В главе 1 я упоминал о том, как APIOps позволяет командам поддерживать упрощенный рабочий поток разработки API, обеспечивает ускоренную обратную связь в ходе дизайн-ревью API и позволяет создавать последовательные API, отвечающие стандартам. Включение артефактов API в пайплайн CI/CD становится центральной реализацией, которая помогает командам воспользоваться этими преимуществами. Пайплайн позволяет организациям быстро и эффективно публиковать API и сопутствующие артефакты, а также обеспечивать соблюдение нужных стандартов соответствия. Он предоставляет централизованный, стандартизированный путь выполнения автоматизированных проверок предлагаемого API, выполнения проверок соответствия и генерации артефактов по определениям API. Когда в руководстве по стилю API, принятом в организации, появляются новые рекомендации, их можно автоматизировать в пайплайне. Таким образом формируется стандартный механизм, который гарантирует, что все изменения в файлах определения API будут проходить через одни и те же автоматизированные проверки качества. В результате сводятся к минимуму ручное выполнение задач управления, риски человеческих ошибок и дефектов, связанных с определениями API.

Я кратко описал преимущества управления артефактами API в пайплайне. Но как это относится к теме APIOps и всего жизненного цикла продукта API? В управлении артефактами в пайплайне развертывания центральное место занимает идея о том, что артефакты API являются важными продуктами разработки, так что к определениям API и документации следует относиться так же, как к коду. В частности, можно публиковать необработанные определения API, чтобы потребители API могли загрузить и использовать их напрямую либо

https://liveinternet.club

воспользоваться ими для генерации документации API или клиентских SDK. Вот некоторые возможные применения определений API:

- *проектирование продукта* — для обсуждений в ходе планирования и проектирования продукта;
- *реализация* — для генерации кода реализации;
- *тестирование* — как основа для тестирования, генерации кода тестов и имитации API;
- *документирование* — для генерации документации API;
- *инструментирование* — для генерации серверных заглушек и клиентских SDK;
- *оценка API* — для генерации метрик соответствия API;
- *мониторинг API* — в инструментировании наблюдаемости и безопасности, а также API-шлюзах, для мониторинга поведения API в развернутых средах.

Из-за этого важно рассматривать определения API как полноправные результаты процесса сборки программного продукта и применять к ним те же инженерные дисциплины, которые обычно применяются к коду.

В этой главе при объяснении концепций пайплайна API я буду исходить из следующего: в организациях действуют сложные требования относительно того, как создавать и развертывать программные продукты и артефакты API, поэтому не существует единого универсального способа управления артефактами PI в пайплайнах CI/CD. Вместо одного рекомендуемого пайплайна я представлю задачу доставки артефактов API, а также опишу, как гипотетическая команда API использует пайплайн CI/CD для управления своими артефактами. Я постараюсь показать, *как* команда подходит к эволюции своего пайплайна, а не представить набор рекомендуемых этапов пайплайна. Помните, что вы можете применить метод A3, описанный в главе 2, для решения вашей конкретной задачи и спроектировать пайплайн для своего контекста. Также я воспользуюсь таблицей рисков, чтобы показать, как пайплайн справляется с рисками управления API на каждом этапе.

Предполагается, что вы уже знакомы с серверами сборки (также называемыми CI/CD-серверами). *Сервер сборки* — платформа, которая автоматизирует все задачи в пайплайне развертывания и объединяет все компоненты. Примеры того, как это делается, я разберу на популярном сервере сборки — платформе GitHub Actions. И вновь я не буду стремиться полностью обучить вас работать с GitHub, а сосредоточусь на примерах того, как команды встраивают определения API и документацию API в свои пайплайны.

Весь код примеров этой главы доступен на страницах https://mng.bz/ngo2 и https://mng.bz/v8a4.

https://liveinternet.club

Глоссарий

Некоторые термины, которые будут использоваться в этой главе.

Команда управления API (API governance team). APIOps решает задачи в масштабе, когда несколько команд занимаются проектированием API и им нужна отдельная команда, отвечающая за управление API, чтобы команды разработчиков могли проектировать и создавать API. В малых организациях с небольшим количеством команд разработки такой проблемы нет, но в крупных она неизбежно встречается. Некоторые крупные организации передают ответственность за управление команде, занимающейся обслуживанием API-шлюза, а в других существует специализированная кроссфункциональная команда платформы API. Какую бы структуру или название вы ни выбрали, в этой книге я буду называть команду, ответственную за контроль над соответствием API, командой управления API.

Предложение по дизайну API (API design proposal), или просто *предложение API*. Следуя подходу design-first, описанному в главе 1, раздел 1.5.1, команды разработки могут отправить предложение API как пул-реквест на ревью команде управления. Предложение API включает предлагаемые изменения API (для RESTful API это будут файлы определений OpenAPI), документацию API и остальную информацию, необходимую команде управления, чтобы разобраться в изменениях и провести дизайн-ревью API.

Документация API (API reference). Представленный в читаемом виде (отрендеренный) файл определения API. Для дизайна OpenAPI на основе REST, согласно спецификации OAS (OpenAPI Specification), определение API представляет собой необработанный файл OpenAPI YAML или JSON. Документация API может читаться человеком и генерируется на основе файла определения API при помощи таких инструментов, как Swagger UI, Stoplight Elements, Redoc или система рендеринга документации API на портале разработчика.

Руководство API (API guide documentation). Этим термином обозначается документация API, отличная от документации, сгенерированной на основе определения OpenAPI. В руководство API включаются страницы документации, представленные на портале разработчика по таким темам, как введение в API, аутентификация, ошибки, пагинация, версии, политики депрекации, учебники и т. д. В этой главе я рассматриваю руководство API как файлы Markdown в репозитории проекта. И хотя для целей этой книги я отношу документирующий контент API либо к документации, либо к руководству API, авторы технических текстов используют более детализированную классификацию. В книге «Docs for Developers: An Engineer's Field Guide to Technical Writing» Джареда Бхатти и др. (Jared Bhatti et al.) (Apress, 2021) авторы делят документирующий контент API на другие типы: README, документация для начинающих, концептуальная документация, процедурная документация, учебники, руководства how-to, справочная документация, справочная документация API и глоссарии.

Сервис размещения кода (code hosting service). Это средства хостинга для исходного кода проекта, документации и других файлов. Создается на основе системы контроля версий (VCS, Version Control System — например, Git), которая ведет историю ревизии файлов, но также предоставляет другие функции управления программными проектами — такие, как системы отслеживания дефектов, документация проектов на базе вики, списки рассылки и средства управления выпуском. Примеры — GitHub, GitLab, Bitbucket и Azure DevOps Server.

https://liveinternet.club

8.1. Трудности совместного использования предложений API

Представьте, что в вашей организации Acme созданием разных внешних API занимаются десять команд. Команды разработки сначала проектируют определения API одновременно с созданием руководств, после чего отправляют определения команде управления для ревью. Как именно они должны отправлять файлы?

Конечно, можно послать их по электронной почте. Команда управления Acme может проанализировать изменения и по электронной почте ответить, одобрены ли отправленные материалы. А может порекомендовать изменения в конкретных строках файлов. Когда команда разработки внесет изменения, она отправит их команде управления, которая сравнит версии файлов и проверит, что изменения соответствуют их рекомендациям. На рис. 8.1 представлен пример диалогов в этом рабочем потоке.

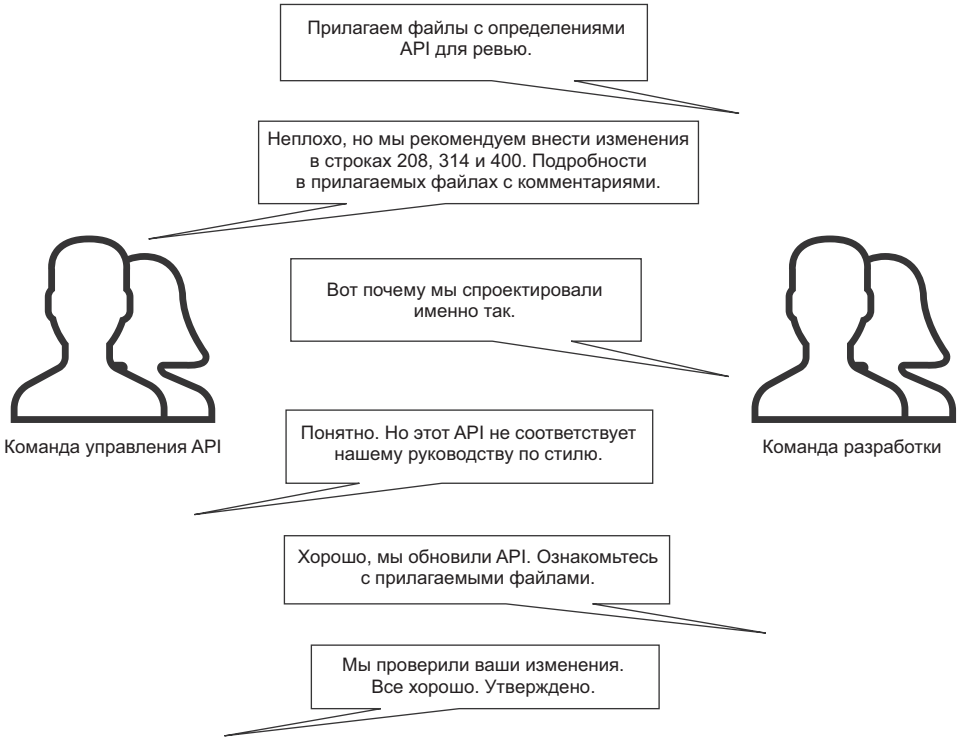


Рис. 8.1. Пример взаимодействия между командой управления и командой разработки в ходе ревью предложения по изменению API

https://liveinternet.club

Обмен по электронной почте быстро усложняется, и за его ходом трудно уследить. В описывамом сценарии ситуация усугубляется тем, что подобная коммуникация может идти параллельно с десятью командами. Основная проблема, с которой сталкиваются команды, — отслеживание изменений в файлах и обсуждение этих изменений.

8.2. Решение: контроль версий

Стандартное решение для организации совместного доступа к изменениям в файлах и совместной работы над ними заключается в хранении файлов определений API в системе контроля версий. В таком случае команда добавляет свои файлы определения API в репозиторий контроля версий своего проекта — так же, как и все остальные исходные файлы. И это касается не только файлов определений API; любые исходные файлы, относящиеся к API (такие, как руководства и тесты соответствия API), также могут храниться в системе контроля версий. Команда разработки может отправить свое предложение API как пул-реквест на ревью командой управления. Прежде чем команда управления займется этим пул-реквестом, он должен пройти автоматизированные проверки; это указывает, что пул-реквест соответствует *минимальным критериям дизайн-ревью* (MDRC, Minimum Design Review Criteria). MDRC — чек-лист качества для проверки того, что предложения API удовлетворяют минимальным требованиям для начала дизайн-ревью. Допустим, в рассматриваемом сценарии команда управления API установила следующие критерии MDRC.

1. В названии пул-реквеста содержится ссылка на рабочий тикет. Предполагается, что в данном случае команда использует Atlassian Jira, а ее рабочие тикеты записываются в формате DEV-*XXXX*, где *X* — цифра. Кроме того, описание пул-реквеста содержит информацию о том, почему необходимо изменение. Я буду называть эти правила *политикой описания пул-реквеста*, то есть набором правил, которым должны соответствовать названия и описания пул-реквестов перед слиянием.
2. Предложение должно проходить все проверки линтера API.
3. Предложение должно проходить все проверки на критические изменения.
4. Предложение должно проходить все автоматизированные проверки *статического анализа текста* в соответствии с руководством API. В частности, проверки расстановки знаков препинания, орфографии, грамматики и стиля.
5. После прохождения MDRC и перед слиянием пул-реквеста с основной веткой репозитория должен быть соблюден пятый критерий — предложение должно пройти ревью и получить одобрение от участника команды управления. Я буду называть эти пять критериев *критериями слияния*.

https://liveinternet.club

8.3. Автоматическая проверка MDRC

В нашем сценарии для первого критерия команда Acme использует скрипт для проверки названия пул-реквеста, как при сборке. Команда также может воспользоваться шаблоном пул-реквеста, чтобы запросить у его создателей описание причин для внесения изменений. Для второго и третьего критериев команда может воспользоваться Spectral (см. главу 3) для проведения проверок линтинга, и Tufin/oasdiff (см. главу 4) для проверки критических изменений при сборке. Задание сервера сборки, которое выполняет эти проверки, называется *сборкой пул-реквеста*. Для четвертого критерия — ревью пул-реквеста участником команды управления Acme — команда может воспользоваться функциональностью «владельцев кода» своего сервиса хостинга, чтобы перед слиянием убедиться, что одобрение от участника команды управления получено. *Владельцы кода* (code owners) — функциональность платформ хостинга кода, которая позволяет командам назначить эксперта предметной области или команду таких экспертов (обычно из другой команды, не владеющей репозиторием) владельцем кода для конкретных файлов, типов файлов или каталогов в репозитории контроля версий. Владелец кода должен одобрить пул-реквест, затрагивающий соответствующие файлы, прежде чем они могут быть слиты. На рис. 8.2 показан примерный процесс автоматической проверки критериев MDRC.

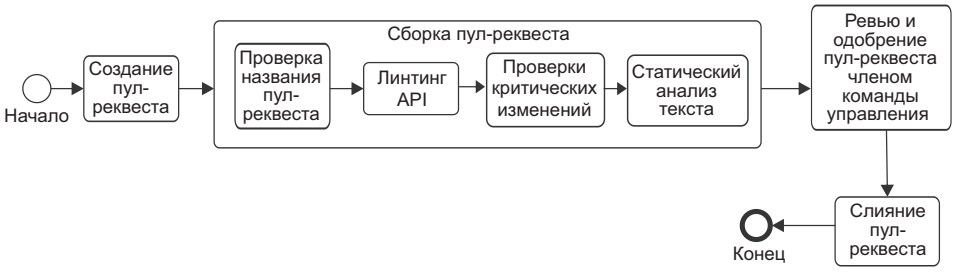


Рис. 8.2. Процесс, используемый командой Acme для проверки того, что пул-реквест удовлетворяет минимальным критериям дизайн-ревью и одобрен командой управления

В этом сценарии необходимо учесть один важный момент: команда Acme уже приняла CI/CD и создала пайплайн развертывания для сборки и развертывания своих программных сервисов в различных средах. Команда не ограничивается простым управлением своими артефактами в этом пайплайне.

Пайплайн развертывания

Пайплайн развертывания состоит из двух основных фаз — *непрерывной интеграции* и *непрерывной доставки* (CI/CD, как уже говорилось). Непрерывная интеграция (CI) — практика, при которой команды регулярно интегрируют свои изменения (не реже

https://liveinternet.club

чем раз в день) с основной веткой репозитория исходного кода. Сервер сборки — платформа, запускающая задачи для преобразования кода в программные артефакты. Он также тестирует программный продукт, чтобы убедиться, что вся программная система продолжает работать после каждого изменения. На этапе CI это технические тесты, ориентированные на разработчиков и позволяющие им быстро проверить, что они ничего не нарушили. Выводом этапа CI является упакованный релиз-кандидат, который затем может быть развернут для тестирования в промежуточных средах.

На этапе непрерывной доставки (CD) релиз-кандидат развертывается и тестируется в средах, напоминающих продакшен (промежуточных средах), чтобы проверить, что он соответствует всем критериям приемлемости, производительности, безопасности и т. д. Развертывания в промежуточных средах автоматизируются и могут инициироваться автоматически или по требованию. Команды, практикующие CD, применяют такие приемы, как флаги функций (feature flags) и добавление пользовательских интерфейсов на последнем шаге, для частичного скрытия функциональности от пользователей.

Непрерывное развертывание — это следующий этап улучшения CD, при котором каждый коммит, проходящий все стадии пайплайна, автоматически развертывается на продакшен без вмешательства человека. Непрерывное развертывание означает, что пользователи видят непрерывный поток улучшений на продакшен, развертывания становятся менее рискованными, а частично построенные функции скрываются при помощи флагов функций.

Я использую следующие термины для обозначения этапов и компонентов пайплайна развертывания.

- *Репозиторий исходного кода* — система контроля версий, используемая для хранения кода, конфигурации и инфраструктурных исходных файлов проекта.
- *Сборка* — автоматизированный процесс компиляции и упаковки кода, а также выполнения тестов, ориентированных на разработчика (например, компонентных и юнит-тестов). Сборка пул-реквеста предоставляет его создателю быструю обратную связь о том, что изменения, внесенные в пул-реквест, работают на техническом уровне. При слиянии пул-реквеста с главной веткой репозитория выполняется сборка главной ветки, и в результате успешный релиз-кандидат публикуется в репозитории артефактов.
- *Система управления зависимостями* — система, хранящая внешние библиотеки и контейнерные образы для внешнего потребления.
- *Репозиторий артефактов* — релизы-кандидаты публикуются в репозитории артефактов. Репозиторий артефактов — версионируемое хранилище двоичных артефактов и связанных с ними метаданных в определенной структуре каталогов. Артефакты сборки хранятся в репозитории артефактов и могут развертываться в различных средах на более поздних этапах пайплайна.
- *Реестр API* — система времени выполнения, которая позволяет пользователям отправлять свои файлы определений API и делиться ими, чтобы потребители API могли искать и просматривать подробную информацию об API.
- *Внеэксплуатационное тестирование* — пайплайн развертывает релиз-кандидат в средах, все более приближенных к продакшен, и тестирует программный про-

https://liveinternet.club

дукт с точки зрения пользователя для повышения уверенности команды в том, что программный продукт готов к выпуску. К числу таких тестов относятся приемочные тесты, тесты производительности и другие, которые организация посчитает необходимыми.

- *Эксплуатационное тестирование* — релизы-кандидаты, прошедшие все внеэксплуатационные проверки, развертываются на продакшен (или его эквиваленте — например, магазине приложений).

Полное обсуждение реализации пайплайна развертывания выходит за рамки этой книги. Ключевой труд по пайплайнам развертывания — «Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation»[1] Дэвида Фарли (David Farley) и Джеза Хамбла (Jez Humble) (Addison-Wesley Professional, 2010). Книга Дэвида Фарли «Continuous Delivery Pipelines: How to Build Better Software Faster» (Dave Farley, 2021) также содержит хорошее введение в тему пайплайнов. Книга Генри ван Мероде (Henry van Merode) «Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines» (Apress, 2023) содержит множество практических рекомендаций по созданию пайплайнов.

Представим, что команда разработки использует пайплайн развертывания, показанный на рис. 8.3. Добавление файлов определений API в репозиторий исходного кода означает, что разработчики могут проверять эти файлы в составе своего пайплайна.

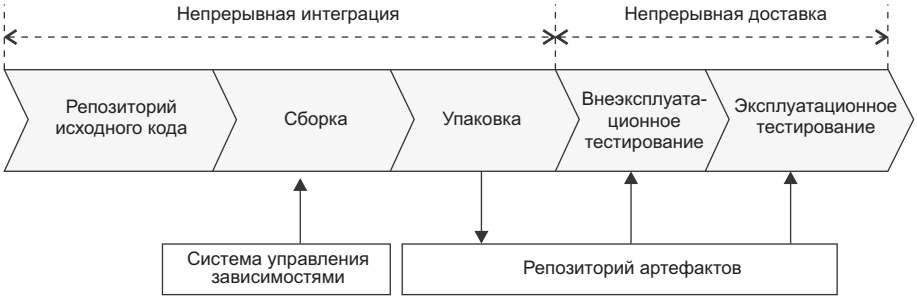


Рис. 8.3. Пайплайн развертывания команды Acme

Я описал высокоуровневое решение для автоматической проверки MDRC в пайплайне. Но прежде чем предлагать вам открыть репозиторий проекта в редакторе Visual Studio Code (VS Code) и поработать с примерами кода для организации проверок слияния в этом сценарии, я хочу познакомить вас с сервером сборки этого примера — GitHub Actions.

[1] Хамбл Д., Фарли Д. «Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ».

https://liveinternet.club

ПРИМЕЧАНИЕ Если вы уже знакомы с GitHub Actions, следующий раздел можно пропустить.

8.4. Введение в GitHub Actions

GitHub Actions (https://github.com/features/actions) — инструмент GitHub для автоматизации рабочих потоков в жизненном цикле разработки ПО, обладающий полной поддержкой CI/CD. В GitHub Actions рабочие потоки активируются, когда происходят конкретные события в репозитории — например, инициируется пул-реквест или слияние ветки. События могут запускать рабочие потоки по одному или нескольким триггерам и настраиваться для выполнения в определенных ветках кода в репозитории. Сами рабочие потоки представляют собой настраиваемые автоматизированные процессы, заданные в файлах YAML, которые размещаются в каталоге `.github/workflows`. Помимо активации на основе событий в репозитории, рабочие потоки можно активировать по расписанию или вручную. Рабочие потоки выполняются на сервере, который называется *исполнителем* (runner). GitHub предоставляет облачные исполнители для Ubuntu Linux, Microsoft Windows и macOS. Вы также можете разместить собственный исполнитель (например, на локальном компьютере), к которому GitHub Actions будет подключаться для выполнения задачи.

Файлы рабочих потоков также содержат *задания*. Задание (job) — набор шагов, выполняемых на одном исполнителе. Шагом может быть либо скрипт командной оболочки, выполняемый на исполнителе, либо действие. *Действие* (action) представляет собой специальное приложение для платформы GitHub Actions для выполнения повторяющихся задач. Например, извлечение кода из репозитория — часто встречающаяся задача в пайплайнах CI/CD, и для нее существует действие: `actions/checkout@v3`. По умолчанию задания в рабочем потоке не имеют зависимостей от других заданий и выполняются параллельно. Однако это поведение можно настроить и изменить. Шаги задания всегда выполняются последовательно и зависят друг от друга.

ПРИМЕЧАНИЕ Дополнительную информацию о платформе GitHub Actions см. в документации по адресу https://docs.github.com/en/actions.

Чтобы увидеть, как работают все эти концепции, рассмотрим следующий пример рабочего потока. Представьте, что у команды разработки Acme имеется сервис Products API, содержащийся в папке `chapter8` проекта. Это проект Java, в котором уже определен файл рабочего потока GitHub Actions. Чтобы найти файл рабочего потока, откройте каталог `.github` в корневом каталоге проекта и перейдите к файлу `.github/workflows/pr-build.yaml`. Рабочий поток сборки пул-реквеста, определенный в `pr-build.yaml`, выполняется при подаче пул-реквеста для главной ветки. Он состоит из четырех шагов, описанных в задании `pr-code-build`:

https://liveinternet.club

- проверка исходного кода действием `checkout/v3`;
- создание виртуальной машины Java (JVM) JDK 17, используемой для выполнения кода, действием `setup-java@v3`;
- компиляция кода с использованием Maven, инструмента автоматизации сборки Java;
- запуск юнит-тестов в проекте с использованием Maven.

На рис. 8.4 показано возможное представление этих шагов в модели GitHub Actions.

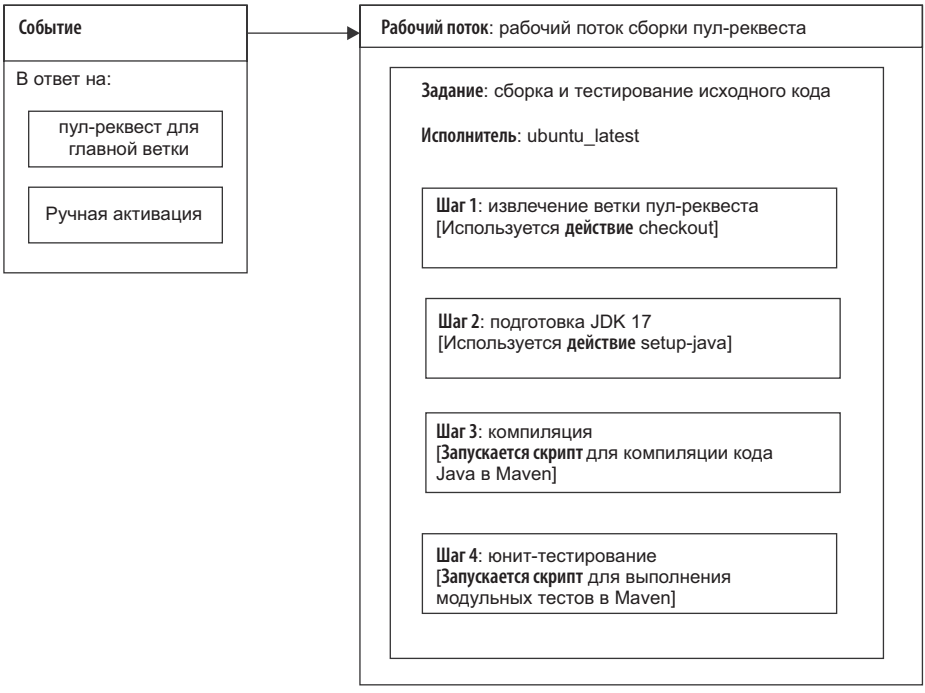


Рис. 8.4. Концепции GitHub Actions, примененные к рабочему потоку сборки пул-реквеста. На платформе GitHub Actions события активируют рабочие потоки, которые выполняются на исполнителях

Код рабочего потока сборки пул-реквеста приведен в листинге 8.1.

Листинг 8.1. Рабочий поток сборки пул-реквеста

```
name: PR build workflow
on:
  pull_request:
    branches: [main]
```

Активирует этот рабочий поток при наличии пул-реквеста для главной ветки

https://liveinternet.club

```
  workflow_dispatch:
jobs:
  pr-code-build:
    name: Исходный код сборки и тестирования
    runs-on: ubuntu-latest
    steps:
      - name: извлечение ветки пул-реквеста
        uses: actions/checkout@v3
      - name: подготовка JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven
      - name: компиляция
        run: |
          cd chapter8
          mvn --batch-mode clean compile
      - name: юнит-тест
        run: |
          cd chapter8
          mvn --batch-mode test
```

Позволяет активировать этот рабочий поток вручную

Задание для сборки кода

Образ исполнителя, размещенный на GitHub и используемый для выполнения рабочего потока

Извлекает код действием actions/checkout@v3

Выполняет команды Maven clean и compile в скрытом пакетном режиме

Начните с выполнения этого рабочего потока. Клонируйте проект, если вы этого еще не сделали, и сохраните его в приватном репозитории GitHub, как описано в приложении Г. Теперь создайте пул-реквест для активации рабочего потока. Это может быть незначительное изменение в любом файле — например, добавление новой строки в конце файла `README.md`. Создайте пул-реквест для этого изменения, чтобы слить свое изменение с главной веткой. После того как вы создадите пул-реквест, должна появиться информация о выполнении проверок пул-реквеста. После краткого введения в GitHub Actions все готово к созданию рабочего потока для выполнения MDRC.

8.5. Подготовка MDRC

Вы уже видели существующее задание `pr-code-build` для компиляции и запуска юнит-тестов в файле рабочего потока `.github/workflows/pr-build.yaml`. Допустим, вы хотите добавить новое задание, которое выполняет проверки MDRC в рабочем потоке. Как это сделать?

Задание должно выполнять четыре проверки, упомянутые в разделе 8.2:

- проверить, что название пул-реквеста содержит ссылку на рабочий тикет;
- выполнить линтинг определения API;
- выполнить проверку критических изменений на основе определения API;
- выполнить статический анализ текста на основе руководства.

На рис. 8.5 показаны эти проверки, смоделированные в рабочем потоке GitHub Actions.

https://liveinternet.club

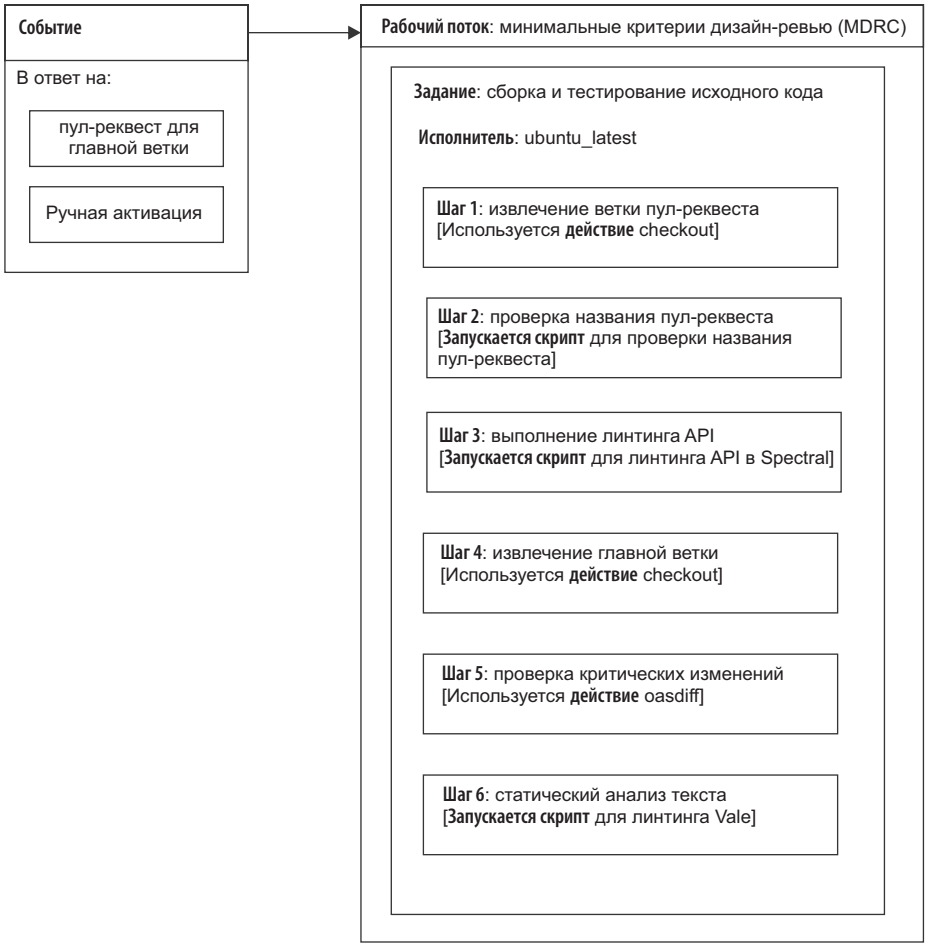


Рис. 8.5. Расширение файла рабочего потока новым заданием для проверки MDRC

Теперь реализуйте шаги GitHub Actions в задании, как описано ниже.

> **ПРИМЕЧАНИЕ** Если вы не хотите самостоятельно создавать пример, а предпочитаете увидеть готовый код, см. файл .github/workflows/pr-build.completed.

8.5.1. Проверка рабочего тикета

Сначала напишите скрипт для проверки первого критерия — что в названии каждого пул-реквеста содержится код рабочего тикета, чтобы ревьюеры пул-реквеста могли найти соответствующий тикет. Скрипт должен получать название пул-реквеста на входе и проверять, что оно содержит код тикета в формате [DEV-*XXXX*], где *X* — цифра. Кроме того, создайте новое задание

https://liveinternet.club

`pr-api-minimum-review-criteria` в файле рабочего потока, которое вызывает ваш скрипт в новом шаге с именем `Check PR title` («Проверка названия пул-реквеста»).

В своей копии проекта перейдите в каталог `chapter8`. Создайте файл с именем `pr-title.sh` и сделайте его исполняемым (в системах на базе Linux/Unix для этого следует выполнить команду `chmod +x ./pr-title.sh`). Добавьте содержимое листинга 8.2 в файл.

Листинг 8.2. Проверка кода рабочего тикета в названии пул-реквеста

```
#!/usr/bin/env bash

pr_title=$1
pattern="DEV-[0-9]{4}"

if [[ ! $pr_title =~ $pattern ]]; then
    echo "Название пул-реквеста: $pr_title"
    echo "Ошибка: название пул-реквеста не содержит
➥ действительный код тикета Jira в формате DEV-XXXX."
    echo "Добавьте код тикета Jira в название пул-реквеста."
    exit 1
fi
```

Регулярное выражение для рабочего тикета, которое должно совпадать с названием пул-реквеста

Если совпадение не найдено, выводится сообщение и возвращается ненулевой код завершения

СОВЕТ Чтобы не вводить код вручную, найдите скрипт `pr-title.completed` в папке `chapter8`; вы можете переименовать его в `pr-title.sh`.

Затем создайте новое задание `pr-api-minimum-review-criteria` в `.github/workflows/pr-build.yaml`. Добавьте новое задание как дочерний узел для узла задания. Сначала задание должно извлекать код проекта в папку `pr-branch` в рабочем каталоге (я объясню, зачем это нужно, позже, когда мы будем писать код проверок критических изменений). Объект контекста GitHub Actions `'${{ github }}'` содержит подсвойство `${{ github.event .pull_request.title }}` с названием пул-реквеста, и его можно передать скрипту. Теперь создайте шаг для выполнения скрипта и передачи этого свойства. В следующем листинге приведен код задания, который можно присоединить (не заменить!) к существующему файлу `pr-build.yaml`.

Листинг 8.3. Задание рабочего потока для минимальных критериев ревью API

```
…
  pr-api-minimum-review-criteria:
    name: Минимальные критерии дизайн-ревью
    runs-on: ubuntu-latest
    steps:
      - name: извлечение ветки пул-реквеста
        uses: actions/checkout@v3
        with:
          path: pr-branch
```

https://liveinternet.club

```
      - name: Проверка названия пул-реквеста
        run: ./pr-branch/chapter8/pr-title.sh
➥ "'${{ github.event.pull_request.title }}'"
```

СОВЕТ Готовый рабочий поток GitHub Actions для этой главы находится в файле `.github/workflows/pr-build.completed` (путь отслеживается от корневой папки проекта).

Когда это будет сделано, создайте пул-реквест. Чтобы убедиться, что код работает, добавьте название пул-реквеста, не содержащее код тикета, и проверьте, что задание завершается с ошибкой. Чтобы тест проходил, отредактируйте название пул-реквеста в GitHub, чтобы оно начиналось с кода тикета — например, `[DEV-1234]`. Обновите пул-реквест незначительным изменением (скажем, добавьте незначащий пробел в любом файле), чтобы увидеть, как задание заново активируется и выполняется. На этот раз оно должно завершиться успешно. Проведите слияние пул-реквеста.

ПРИМЕЧАНИЕ Мы рассмотрели очень простой сценарий. Конечно, его можно расширить, чтобы он проверял тикет Jira с использованием REST API Jira.

8.5.2. Создание шаблона пул-реквеста

Для выполнения первого критерия MDRC — наличия у пул-реквеста полезного описания — самое время создать шаблон пул-реквеста, который будет подсказывать его создателям, какую информацию им нужно предоставить.

В папке `.github/` создайте файл `pull_request_template.md`, содержащий следующий фрагмент:

```
## Зачем нужно это изменение?

## Каков масштаб этого изменения?

## Чек-лист
- [ ] Запускались ли эти тесты локально?
```

СОВЕТ Также можно скопировать код из файла `pull_request_template.completed` в том же каталоге.

Сохраните код и создайте пул-реквест. В дальнейшем шаблон должен появляться при создании новых пул-реквестов.

8.5.3. Выполнение линтинга API

Теперь создадим автоматизированную проверку второго критерия MDRC — пул-реквест должен проходить все проверки линтинга для API. Для этого добавьте в задание `pr-api-minimum-review-criteria` новый шаг для запуска линтинга

https://liveinternet.club

API. Исполнитель `ubuntu-latest` включает установленный экземпляр Node, но в Spectral CLI такого нет. Вместо установки Spectral можно просто выполнить его командой `npx`. Она позволяет легко установить и запустить исполняемый файл `Node.js` из реестра NPM. Следующий фрагмент кода показывает, как добавить новый шаг с запуском линтинга Spectral на основе определения API, используя файл с набором правил `json-api.ruleset.yaml`. Добавьте его непосредственно после шага названия Check PR («Проверка пул-реквеста»):

```
- name: Запуск линтинга API
  run: |
    npx @stoplight/spectral-cli lint
➥ "./pr-branch/chapter8/apis/product-catalog.oas.yaml"
➥ --ruleset "./pr-branch/chapter8/apis/rulesets/json-api.ruleset.yaml"
```

Сохраните код, создайте пул-реквест (не забудьте включить код-пустышку тикета в название!) и понаблюдайте за ходом сборки пул-реквеста. Вывод в журналах должен завершаться следующим сообщением:

```
No results with a severity of 'error' found![1]
```

> **СОВЕТ** В этом примере шаг линтинга запускался как скрипт из командной строки исполнителя. Возможен и другой вариант — использование действия Spectral GitHub (https://github.com/stoplightio/spectral-action).

8.5.4. Выполнение проверок критических изменений

Для третьего критерия MDRC — прохождения пул-реквестом всех проверок критических изменений — создайте в задании `pr-api-minimum-review-criteria` еще один шаг. На нем для проверки критических изменений должно использоваться действие `oasdiff/oasdiff-action/check-breaking@main` (см. документацию на странице https://github.com/oasdiff/oasdiff-action). Но сначала необходимо извлечь исходный код из главной ветки в папку main в рабочем каталоге исполнителя, чтобы сравнить файл определения API в папке main с файлом из папки pr-branch. Следующий фрагмент показывает, как это делается:

```
- name: Извлечение главной ветки
  uses: actions/checkout@v3
  with:
    ref: main
    path: base
```

Главная ветка извлекается в базовый каталог

Затем добавьте шаг для выполнения проверки критических изменений:

```
- name: проверка критических изменений
  uses: oasdiff/oasdiff-action/check-breaking@main
  with:
```

[1] No results with a severity of 'error' found! — Результатов уровня 'error' не найдено. — *Примеч. пер.*

https://liveinternet.club

```
base: './base/chapter8/apis/product-catalog.oas.yaml'
revision: './pr-branch/chapter8/apis/product-catalog.oas.yaml'
fail-on-diff: true
```

Зафиксируйте этот код и создайте пул-реквест для его проверки. Затем убедитесь, что он работает: внесите критическое изменение в файл `chapter8/apis/product-catalog.oas.yaml`, заменив путь `/v1/catalog/categories` на `/vx/catalog/categories` (фактически удалите один путь и добавьте новый). Зафиксируйте изменение и создайте пул-реквест. На этот раз сборка пул-реквеста должна завершиться неудачей. В журнале должна появиться примерно такая запись об ошибке:

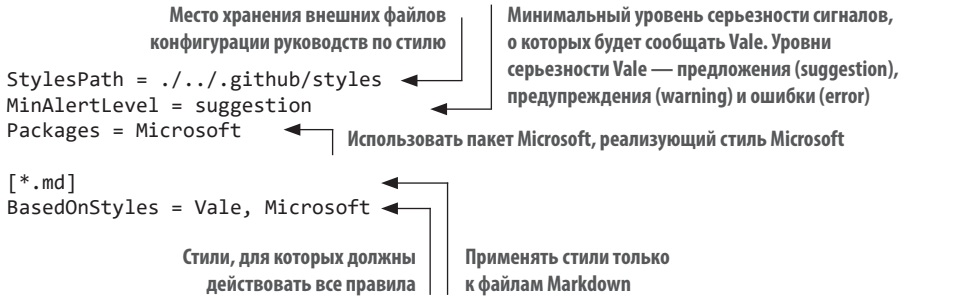
```
error at original_source=./base/chapter8/apis/product-catalog.oas.yaml,
➥ in API GET /v1/catalog/categories api path removed without
➥ deprecation [api-path-removed-without-deprecation][1].
```

Создайте новый коммит, чтобы отменить свои изменения.

8.5.5. Статический анализ текста

Добавьте шаг для проверки четвертого критерия MDRC — прохождения пул-реквестом всех проверок пунктуации, орфографии и грамматики. Vale (https://vale.sh) — открытый статический анализатор с интерфейсом командной строки для проверки документации на соответствие стилевым руководствам. Используйте Vale в задании `pr-api-minimum-review-criteria`, как описано ниже.

В папке `chapter8` вашего примера проекта присутствует файл `.vale.ini`, который содержит конфигурации по умолчанию для статического анализатора Vale. `.vale.ini` в этом проекте настроен для загрузки и использования внешних файлов конфигурации руководств по стилю, основанных на Microsoft Writing Style Guide (https://learn.microsoft.com/en-us/style-guide). Они оформлены в виде пакетов (package), содержащих общую конфигурацию Vale. Пакеты Vale доступны по адресу https://vale.sh/hub. Конфигурация в файле Vale выглядит так:



[1] Ошибка в original_source=./base/chapter8/apis/product-catalog.oas.yaml, в API GET /v1/catalog/categories путь API удален без предупреждения о депрекации [api-path-removed-without-deprecation]. — *Примеч. пер.*

https://liveinternet.club

Руководство API для проекта содержится в папке `chapter8/apis/docs`. Добавьте шаг для загрузки, установки и выполнения статического анализатора Vale с этой конфигурацией:

```
      - name: Статический анализ текста
        run: |
          wget -nv
➥ https://github.com/errata-ai/vale/releases/download/v2.15.4/
➥ vale_2.15.4_Linux_64-bit.tar.gz
          mkdir bin && tar -xvzf vale_2.15.4_Linux_64-bit.tar.gz -C bin
          export PATH=./bin:"$PATH"
          cd pr-branch/chapter8
          vale sync --config=
➥ 'pr-branch/chapter8/.vale.ini'
          vale pr-branch/chapter8/apis/docs
➥ --config='pr-branch/chapter8/.vale.ini'
```

Применяет vale ко всем документам в каталоге apis/docs с использованием конфигурации .vale.ini

Загружает и устанавливает внешние пакеты с файлами конфигурации стилей (так, как указано в .vale.ini) и сохраняет их в каталоге styles

Зафиксируйте код и создайте пул-реквест с этим шагом. При этом должно активироваться выполнение рабочего потока `pr-build`, а новый шаг выведет в журнал сообщение, показанное на рис. 8.6.

```
24
25  pr-branch/chapter8/apis/docs/introduction.md
26  1:29  suggestion  'REST' has no definition.       Microsoft.Acronyms
27  2:32  suggestion  Use the Oxford comma in        Microsoft.OxfordComma
28                    'codes, authentication and
29                    verbs.'.
30  2:80  suggestion  'REST' has no definition.       Microsoft.Acronyms
31
32  ✔ 0 errors, 0 warnings and 3 suggestions in 5 files.
```

Рис. 8.6. Вывод при запуске Vale для руководства API

> **СОВЕТ** В рассмотренном примере шаг статического анализа текста запускается вызовом Vale из командной строки исполнителя. То же самое можно сделать другим способом: воспользоваться действием GitHub Actions для Vale (https://github.com/errata-ai/vale-action).

8.5.6. Запрос одобрения пул-реквеста от команды управления

Предположим, нам необходимо, чтобы все изменения в файлах определений API или руководствах были одобрены участником команды управления API. В этом случае нам понадобится коллега с учетной записью GitHub, который станет ревьюером из команды управления API. Он будет играть роль владельца

https://liveinternet.club

кода — эксперта предметной области, который должен провести ревью пул-реквеста, изменяющего заданные файлы перед слиянием.

> **ПРИМЕЧАНИЕ** Если у вас есть дополнительная учетная запись GitHub, вы также можете воспользоваться ею в этом примере.

Для этого необходимо обеспечить соблюдение следующих требований к главной ветке.

- Все изменения должны вноситься через пул-реквест.
- Все изменения в файлах и папках в каталоге `chapter8/apis` требуют одобрения от коллеги.
- Для всех пул-реквестов должны запускаться и проходить задания `pr-code-build` и `pr-api-minimum-review-criteria`. Иначе говоря, статус запуска всех этих заданий должен быть успешным.

> **ПРИМЕЧАНИЕ** Для проверок статуса CI-сборка на платформе GitHub Actions использует функцию статусов коммита GitHub REST API, чтобы помечать коммиты состояниями `error` (ошибка), `failure` (сбой), `pending` (ожидание) или `success` (успех). API также позволяет предоставить необязательное описание, содержащее краткую сводку по сборки. Эти состояния отображаются в интерфейсе GitHub на пул-реквестах, содержащих соответствующие коммиты. Подробнее о статусах GitHub см. на https://docs.github.com/en/rest/commits/statuses.

Чтобы обеспечить соблюдение указанных выше условий, создайте в корневом каталоге проекта файл `CODEOWNERS`, в котором ваш коллега будет указан как владелец кода для каталога `chapter8/apis/directory`. Для этого можно переименовать файл `_CODEOWNERS` в `CODEOWNERS` и включить в него идентификатор GitHub вашего коллеги. Сохраните файл в репозитории, создайте пул-реквест и слейте его с главной веткой.

> **ПРИМЕЧАНИЕ** В этом примере мы рассматриваем одного ревьюера. Однако GitHub также поддерживает возможность назначения в качестве владельца кода целой команды. Подробнее об этой возможности см. на https://mng.bz/4JGj.

Затем добавьте своего коллегу в участники проекта. Перейдите к настройкам репозитория, щелкните на Collaborators (Участники) и Add People (Добавить участников). Найдите своего коллегу по идентификатору пользователя GitHub и добавьте его.

Теперь настройте правила защиты ветки GitHub, чтобы объявить, что все изменения осуществляются через пул-реквест, изменения должны быть одобрены владельцами кода, а проверки статуса сборок веток должны проходить успешно до слияния пул-реквеста с главной веткой. Для этого перейдите на вкладку Settings (Настройки) своего репозитория GitHub. Выберите раздел Branches

https://liveinternet.club

(Ветки) на левой панели, а затем щелкните на кнопке Add Branch Protection Rule (Добавить правило защиты ветки). В поле Branch Name Pattern (Шаблон имени ветки) введите имя главной ветки `Main`. Затем установите флажки Require Status Checks to Pass before Merging (Требуется успешная проверка статуса перед слиянием) и Require Review from Code Owners (Требуется ревью владельцами кода). В поле поиска найдите и выберите Minimum API Design Review Criteria and Build and Test Source Code (Минимальные критерии дизайн-ревью API и сборка и тестирование исходного кода) (рис. 8.7).

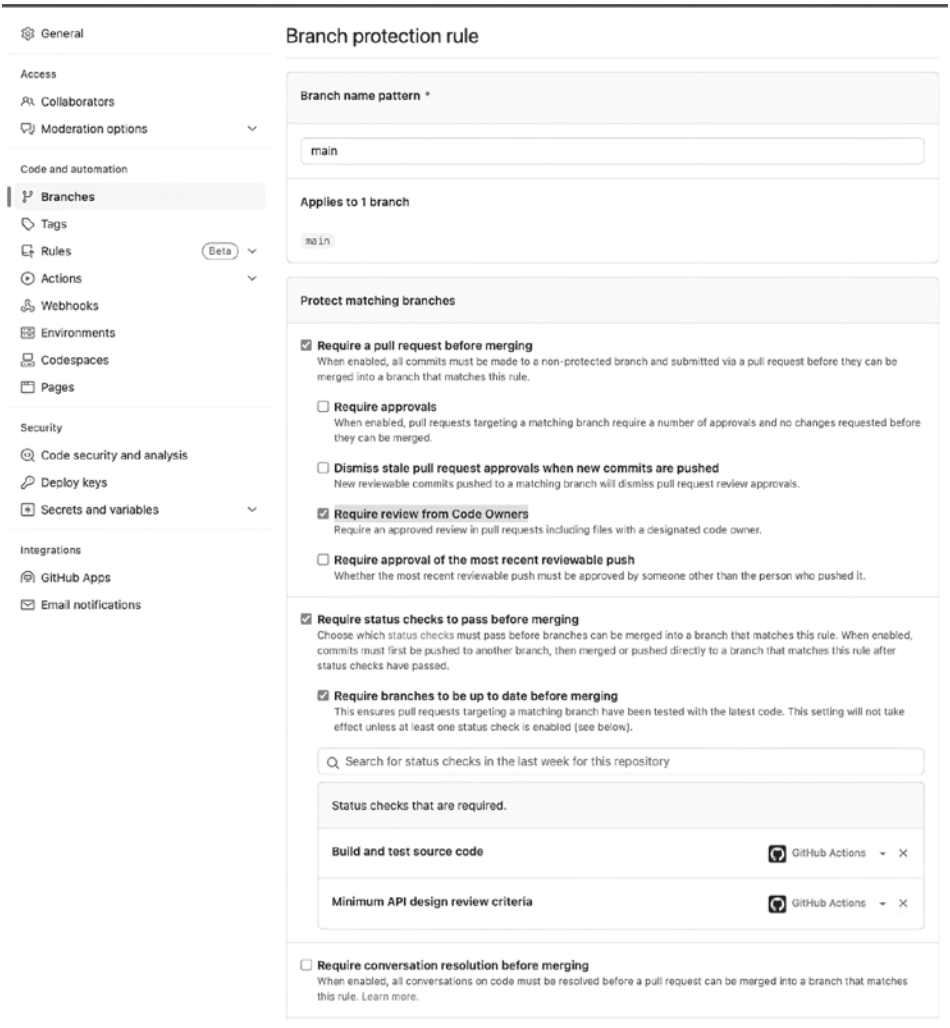


Рис. 8.7. Настройка правил защиты веток на GitHub

https://liveinternet.club

Теперь проверьте, что правило владельца кода работает. Обновите любой файл Markdown в каталоге `chapter8/apis/directory`. Сохраните изменения и создайте пул-реквест. Должно появиться оповещение о том, что для пул-реквеста требуется ревью владельцем кода (рис. 8.8).

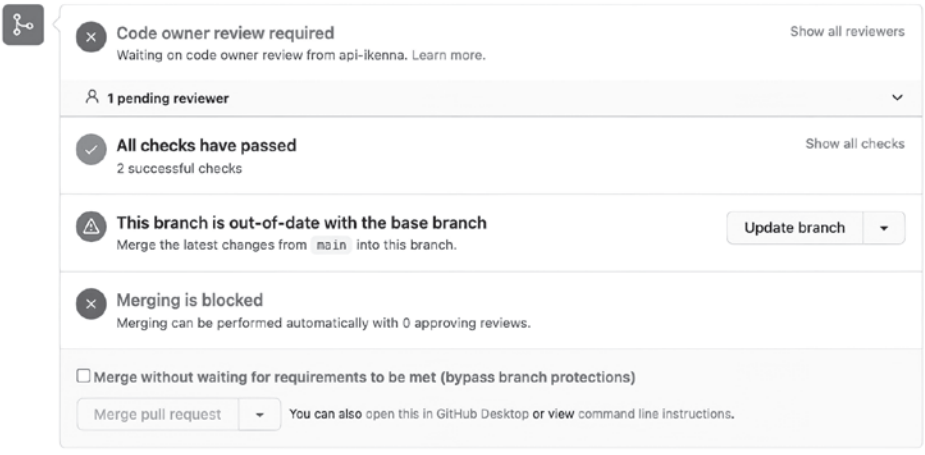


Рис. 8.8. Для пул-реквестов, изменяющих файлы в каталоге chapter8/apis, необходимо ревью владельцем кода

Вы реализовали все правила MDRC, которые используются для контроля на этом этапе пайплайна развертывания. В следующем разделе рассмотрим риски, которые эти средства контроля позволяют снизить в данном примере.

8.6. Автоматизированный контроль управления

Проверки качества сборки пул-реквестов необходимы, чтобы убедиться, что на ревью команде управления API поступят только те предложения, которые прошли необходимые проверки. Проверки качества — средства, которые проверяют, что входные данные этапа пайплайна развертывания удовлетворяют определенным критериям; только после этого можно переходить на следующую стадию. На каждом этапе пайплайна развертывания существуют входные и выходные данные, а также *акторы*, *действия*, *риски* и *контрольные меры*.

- *Риск* — вероятность ущерба или отрицательных последствий, обусловленных уязвимостями, которых можно было избежать.
- *Контрольные меры* — действия, предпринимаемые для устранения рисков.
- *Актор* — роль, выполняющая задачи на этапе пайплайна.
- *Действие* — задача, выполняемая акторами на этапе пайплайна.

https://liveinternet.club

Это структурные элементы модели управления, которая помогает командам управления определять автоматизированные контрольные меры на каждом этапе.

СОВЕТ Подробнее об этой модели автоматизированного управления см. в публикации «DevOps Automated Governance Reference Architecture: Attestation of the Integrity of Assets In the Delivery Pipeline» форума DevOps Enterprise Forum (IT Revolution, 2019).

Таблица 8.1 содержит описание этих элементов на этапе управления исходным кодом предложений по дизайну API (для сценария, который мы разбираем). Обратите внимание, что на более общем уровне эта модель может применяться к изменениям кода в репозитории, но здесь я ограничиваю ее область действия только предложениями по дизайну API.

Таблица 8.1. Модель управления для предложений по дизайну API на этапе исходного кода пайплайна развертывания

Элемент	**Подробности**		
Цель	Обеспечивает быструю обратную связь для разработчиков API по объективным проверкам соответствия стилю API		
Входные данные	Предложение по дизайну API: • файлы определения OpenAPI; • проекты справочной документации API в формате Markdown; • номера рабочих тикетов		
Выходные данные	Предложение по дизайну API, отвечающее минимальным критериям ревью проекта: • отсутствуют ошибки линтинга API уровня error; • отсутствуют критические изменения; • отсутствуют ошибки документации API уровня error		
Акторы	• Дизайнер API; • команда управления API		
Триггер	Создание пул-реквеста		
Действия	• Создать пул-реквест; • ревью, утверждение и слияние пул-реквестов		
Риски и контрольные меры	Риск	Устранение	Контрольные меры
	Соответствие API	Проверьте, что определения OpenAPI соответствуют руководству по стилю, принятому в организации	• Линтинг API сборки пул-реквеста • Одобрение владельцев кода

https://liveinternet.club

Риски и контрольные меры	Риск	Устранение	Контрольные меры
	Критические изменения	Проверьте, что изменения в дизайне API не создают незапланированных критических изменений	Проверки критических изменений для сборки пул-реквеста
	Трудности с чтением документации API	Проверьте руководство API на предмет пунктуации и грамматических ошибок	Статический анализ текста в сборке пул-реквеста
	Недостаточная отслеживаемость изменений функциональности	Предоставьте функциональность продукта и метаданные информации выпуска, необходимые для контекста рецензирования пул-реквеста	Проверки политики описания пул-реквеста

Здесь представлены контрольные меры для примера, рассматриваемого в книге. Для вашего проекта некоторые из них могут оказаться неподходящими, или вам могут понадобиться другие. Например, вам необходимо проверить, что отправленные определения API успешно используются для генерации клиентских SDK. Эта таблица поможет вам продумать, что лучше всего подходит для вашего сценария.

Линтинг API и проверки критических изменений рассматривались в предыдущих главах, поэтому я не буду повторяться, но мы обсудим владельцев кода, политики описания пул-реквеста и статический анализ текста. Но для начала стоит обсудить сборку пул-реквеста в целом.

8.7. Сборки пул-реквеста

Пул-реквесты упрощают совместную работу проектировщиков и ревьюеров над изменениями. Пул-реквесты оповещают ревьюеров о необходимости проверить изменение, а также предоставляют удобный пользовательский интерфейс для обсуждения предлагаемых изменений, прежде чем интегрировать их в ветку. Этот пользовательский интерфейс предоставляет эффективные средства для отслеживания обсуждений и последующих изменений, связанных с пул-реквестом. Возможность ссылаться на пул-реквест, в котором было введено изменение, полезна для отслеживания проблем. Но прежде чем рецензенты приступят к ревью пул-реквеста, они должны убедиться, что все необходимые автоматизированные проверки командной строки (линтинг API, проверки критических изменений и другие тесты) были выполнены и прошли успешно. Как это сделать?

https://liveinternet.club

При создании пул-реквеста сервер сборки также можно настроить на автоматический запуск задания для его проверки. Команды используют такие задания для пул-реквеста, чтобы проверить, компилируется ли код в ветке пул-реквеста, проходят ли проверки статического анализа и юнит-тесты. По завершении сборка пул-реквеста отображает статус задания сборки в пользовательском интерфейсе пул-реквеста, что позволяет как отправителю, так и ревьюерам легко видеть результат. Если сборка завершается неудачей, отправитель пул-реквеста может перейти из пул-реквеста к результатам задания сборки, чтобы просмотреть журналы и выяснить причину сбоя.

Сборки пул-реквеста также могут применяться к предложениям по дизайну API. Линтинг API может выполняться с файлами определений API, а сборку можно настроить на завершение с ошибкой при возникновении проблем линтинга уровня ERROR. Проверки критических изменений могут работать на основе новых и старых версий файлов определений, а сборка должна завершаться сбоем при критических изменениях. Для руководств API в файлах Markdown сборка может завершиться неудачей в зависимости от настроенного уровня проблем линтинга. Если сборка пул-реквеста проходит все проверки, команда управления API может начинать ревью пул-реквеста, зная, что критерии MDRC соблюдены. Но ей все равно придется проверить отчет сборки, чтобы найти проблемы, обнаруженные при сборке (например, проблемы линтинга уровня WARNING), которые не привели к ее сбою. Конечно, также можно составлять скрипты, выполняющие любые специальные проверки, которые могут понадобиться.

У некоторых сервисов хостинга кода и серверов сборки могут существовать предварительно интегрированные сборки пул-реквеста, как было показано в примере с GitHub и GitHub Actions. Но в зависимости от выбранного провайдера вам, возможно, придется создать вебхук на платформе размещения кода, чтобы она могла совершать вызов сервера сборки при возникновении события пул-реквеста.

Многие разработчики привыкли использовать пул-реквесты и их сборки, создавая много запросов за день. Поэтому их внедрение для дизайн-ревью API не потребует значительных изменений рабочего процесса и специальной подготовки.

8.8. Политики описания пул-реквеста

Политики описания пул-реквеста представляют собой набор правил, определяющих, какую информацию должны содержать качественные названия и описания пул-реквеста, чтобы ревьюер понимал, что содержит пул-реквест и почему он создан. Отправитель пул-реквеста не всегда доступен, поэтому пул-реквест должен предоставлять полный контекст; в противном случае ревьюеру придется искать ответы на свои вопросы самостоятельно. Например, команда управления API может решить, что пул-реквест предложения API должен включать код рабочего тикета, объясняющего функциональность, для которой создавался дизайн API. Включение рабочего тикета в название или описание пул-реквеста

https://liveinternet.club

оправданно, если тикет предоставляет полезную информацию для ревьюера. Если же соответствующий тикет пуст или не содержит необходимой информации, это замедлит работу ревьюеров, так как им придется самостоятельно искать ее или строить догадки.

Шаблон пул-реквеста напомнит отправителям о необходимости предоставить эту информацию. Шаблон дает возможность настроить и стандартизировать информацию, переданную отправителями пул-реквеста в его описании. Его можно сравнить с чек-листом того, что должен проверить и указать отправитель пул-реквеста. Шаблоны обычно создаются в виде файлов Markdown в системе размещения кода. Например, в GitHub для этого в репозитории создается файл `pull_request_template.md`. Некоторые системы хостинга позволяют создавать разные шаблоны пул-реквестов для разных веток. Другие обеспечивают проверку содержимого шаблона — например, что нужный флажок установлен.

Хотя шаблоны пул-реквестов – это полезная подсказка для их отправителей, напоминающая им о необходимости предоставить контент для ревьюеров, шаблоны не проверяют введенную информацию. Из-за этого отправители пул-реквестов могут их просто игнорировать. Кроме того, чтобы стимулировать отправителей пул-реквестов к заполнению шаблона, полезно оставить в нем минимум вопросов. GitHub предоставляет хорошую подборку примеров шаблонов пул-реквестов по адресу https://mng.bz/X1ma.

Политики коммита

Очевидно, политики описаний пул-реквеста работают на уровне пул-реквеста. Однако проверка названия и описания изменений может также происходить на уровне коммита. Я использую термин «политика коммита» для обозначения набора автоматизированных правил, соответствие которым может проверяться для каждого нового коммита в VCS. Коммиты принимаются, если они проходят проверку соответствия правилам, и отклоняются в остальных случаях; обычно отклонение сопровождается сообщением, объясняющим его причины. Политики коммита могут использоваться для обеспечения отслеживаемости между исходным кодом в отдельных коммитах и связанных с ним рабочими тикетами или пользовательскими историями. При таком использовании политики коммита помогают ускорить дизайн-ревью API на основе пул-реквеста, так как команда управления может организовать другие проверки, которые посчитает уместными. Политики коммита важны для GitOps и APIOps, где VCS является одновременно источником истины и центральным контрольным механизмом проекта.

В зависимости от настройки политики коммита могут применяться до его создания, чтобы избежать ошибочных коммитов. Это поддерживает механизм быстрого отказа (fail-fast) без ожидания проверки пул-реквеста сервером сборки и не замедляет работу отправителя пул-реквеста. Политики коммита также могут выполняться на сервере при отправке коммитов. В Git политики коммита могут определяться при помощи хуков Git на стороне клиента или сервера. *Хук* Git — это настраиваемый скрипт, который выполняется автоматически при возникновении события в репозитории

https://liveinternet.club

Git. Например, чтобы определить хук Git для проверки того, что коммит начинается с обозначения тикета JIRA в формате *XXX-YYYY*, где *X* — прописная буква, а *Y* — цифра, можно включить следующее определение в исполняемый файл с именем `pre-commit` в папке `.git/hooks` репозитория Git проекта:

```
#!/bin/bash

commit_msg_file=$1

# Чтение файла сообщения коммита

commit_msg=$(cat "$commit_msg_file")

# Шаблон содержащий код тикета Jira (DEV-XXXX)
Pattern="DEV-[0-9]{4}"

# Проверка того, что сообщение коммита не содержит кода тикета Jira
if [[ ! $commit_msg =~ $pattern ]]; then
    echo "Ошибка: сообщение коммита не содержит действительный
➥ номер тикета Jira в формате DEV-XXXX."
    echo "Добавьте номер тикета Jira в сообщение коммита."
    exit 1
fi
```

Хуки Git должны быть быстрыми, надежными и хорошо документированными. Они не должны изменять файлы без согласия автора.

Проблема с хуками Git на стороне клиента заключается в том, что они должны создаваться на рабочей станции каждого разработчика, а не простым клонированием репозитория. Существуют разные способы обойти это ограничение, один из которых — использование пакетного менеджера хуков, в том числе с открытым исходным кодом, например pre-commit (https://pre-commit.com), который управляет установкой и выполнением хуков.

Некоторые сервисы хостинга кода имеют продвинутые возможности или плагины, предоставляющие упрощенные средства настройки политик коммита через веб-интерфейс. Примеры плагинов политик коммита для систем хостинга кода — Better Commit Policy for Bitbucket (https://mng.bz/y8pd) и Control Freak for Bitbucket (https://mng.bz/MZrB).

Для проверки описаний сообщений коммита также можно воспользоваться такими CLI-инструментами с открытым исходным кодом, как commitlint (https://github.com/conventional-changelog/commitlint) и Git Lint (https://github.com/bkuhlmann/git-lint).

8.9. Владельцы кода

Файл определения OpenAPI для внешнего API может находиться в одном репозитории с исходным кодом сервиса API (как в нашем примере) или в отдельном репозитории, предназначенном только для файлов определений API. Если команда, ответственная за обновление дизайна API, обладает разрешениями на запись

https://liveinternet.club

и одобрение пул-реквестов для репозитория, она может одобрить пул-реквест на изменение API самостоятельно, не привлекая к ревью команду управления. Это может стать проблемой, поскольку в крупном проекте команда управления API отвечает за целостность дизайна API в пределах организации, и в нее входят отраслевые эксперты по стандартам API. Если команда разработки не подключает к ревью команду управления, в дизайне могут возникнуть несоответствия, не покрываемые линтингом API и проверкой критических изменений. Может быть упущена критическая обратная связь, которую бы могла предоставить команда управления API. Но как гарантировать, что каждый пул-реквест поступит на ревью к команде управления?

Можно назначить команду управления API *владельцем кода* для файлов определения и документации API. Назначение владельцев кода для конкретных файлов, типов файлов или каталогов в репозитории контроля версий означает, что конкретные люди или команды, указанные как владельцы кода, должны одобрять любые пул-реквесты, относящиеся к этим файлам; только после этого слияние пул-реквеста станет возможным. И владельцы кода получают оповещения каждый раз, когда создаются пул-реквесты, требующие их внимания.

Чтобы назначить владельцев кода в хостинге кода на основе Git, поддерживающего этот контрольный механизм, добавьте файл `CODEOWNERS` в корневой каталог репозитория исходного кода. Учтите, что в некоторых системах хостинга кода для использования этой возможности может потребоваться плагин или дополнение.

`CODEOWNERS` — полезный механизм подключения команды управления API к ревью, но будьте внимательны при его реализации. Если обновление репозитория будет требовать одобрения со стороны многих владельцев кода за пределами базовой команды, это может замедлить разработку. А это противоречит цели APIOps по минимизации препятствий в ходе разработки для ускорения проектирования и доставки API при одновременном соблюдении стандартов. Я рекомендую ограничиться минимальным количеством владельцев кода для конкретного определения API, достаточным для заданного уровня внешнего управления. Для большинства других изменений ревью пул-реквеста желательно проводить внутри команды, вносящей изменения, потому что коммуникация в пределах команды проходит быстрее.

Вы можете назначить команду управления API владельцами кода для файлов определений API. Это гарантирует, что участники этой команды как эксперты по стандартам API должны будут проанализировать и одобрить изменения в файлах определений API, прежде чем интегрировать их в главную ветку.

На GitHub имеется страница, объясняющая, как настроить владельцев кода (https://mng.bz/4JGj); такая же страница есть и в GitLab (https://docs.gitlab.com/ee/user/project/codeowners/). Mibix Software предлагает плагин Bitbucket Code Owners (https://mng.bz/aEpm).

https://liveinternet.club

ПРИМЕЧАНИЕ В этой книге я различаю внутренние API (например, API микросервисов, принадлежащие команде, которые должны использоваться другими микросервисами, используемыми той же командой) и внешние (доступ к которым должен предоставляться за пределы организации или организационного подразделения и которыми необходимо управлять как продуктом с входными данными от многих стейкхолдеров). Команды управления API обычно имеют дело со вторыми, особенно на этапе проектирования. Именно здесь полезно назначать владельцев кода, поскольку команда управления API должна одобрить дизайн. При этом необходимо задать себе важный вопрос: кому стоит доверить ревью этого изменения? Для команд управления API основной сферой ответственности становится ревью изменений дизайна API, обращенных вовне.

8.10. Статический анализ текста

Составление руководств API подразумевает написание технического текста, соответствующего требованиям к языку и стилю изложения. Без проверки авторы могут допускать разные ошибки: орфографические, грамматические и пунктуационные. Также они могут включать лишние слова, жаргонизмы, клише и неидиоматические выражения. Все это усложняет чтение руководств API.

Статические анализаторы текста, или линтеры, решают подобную проблему. Эти программы сканируют и анализируют документы, выявляя ошибки синтаксиса и стиля на основании экспертных рекомендаций. В отличие от простых программ проверки грамматики они могут поддерживать настройку правил и стилей (как и линтеры API). Это позволяет авторам писать тексты, соответствующие стилю и тональности, принятым в конкретной организации. Примеры линтеров — Vale (https://vale.sh), textlint (https://textlint.github.io/) и proselint (https://github.com/amperser/proselint). Они могут выполняться как инструменты командной строки или же использоваться как плагины редактора (у всех этих программ существуют плагины для VS Code).

Vale — популярный и быстрый статический анализатор текстов. В его файле конфигурации (по умолчанию файл `.vale.ini` находится в папке, из которой он был запущен) хранится руководство по стилю языка, которое должно использоваться программой. Vale работает с форматами Markdown, AsciiDoc и reStructuredText.

СОВЕТ Чтобы узнать больше о статических анализаторах текстов, рекомендую прочитать статью «Linting Science Prose and the Science of Prose Linting» от авторов proselint, Майкла Д. Пэйсера (Michael D. Pacer) и Джордана У. Сухоу (Jordan W. Suchow) (https://mng.bz/gvKv). Дополнительную информацию о том, как работает Vale, см. в посте Джозефа Като (Joseph Kato) «Introducing Vale, an NLP-Powered Linter for Prose» на странице https://mng.bz/eozV.

Textlint — еще один популярный линтер, написанный на JavaScript. Он загружает правила в формат плагинов и может проверять тексты в формате Markdown,

https://liveinternet.club

простой текст и файлы HTML. Наконец, proselint — популярный инструмент на базе Python. Его цель — точно применять руководства по стилю от авторитетных писателей и редакторов.

Следуя принципу APIOps, согласно которому по возможности необходимо хранить все исходные файлы проекта в системе контроля версий, я рекомендую оформлять руководства API в виде файлов Markdown, хранящихся в репозитории исходного кода проекта. Это позволит использовать линтеры для их проверки в пайплайне доставки. В пайплайн развертывания можно включить инструменты, отправляющие файлы на портал разработчика. CI-инструменты портала разработчика — такие, как Readme CLI (https://github.com/readmeio/rdme), — позволяют отправить файлы на портал разработчика Readme. Впрочем, вы можете создать собственные инструменты, если используете кастомную систему портала разработчика.

Документация как код

Статический анализ текста руководств API связан с концепцией рассмотрения технической документации как кода (Docs as Code). В книге «Docs Like Code: Collaborate and Automate to Improve Technical Documentation» (3-е изд., Just Write Click, 2022) Энн Джентл (Anne Gentle) описывает основные принципы этого подхода к документированию API.

- Храните исходные файлы документации в системах контроля версий. Документация должна храниться в облегченных форматах разметки текста — таких, как Markdown, reStructuredText и AsciiDoc.
- Артефакты документации должны создаваться на основе исходных файлов автоматически.
- Выполняйте автоматизированные тесты документации.
- По аналогии с код-ревью проводите ревью изменений в документации.
- Публикуйте артефакты документации с минимумом вмешательства со стороны человека.

В рамках подхода «документация как код» рабочий поток документации повторяет рабочий поток, применяемый командой разработки. Это позволяет командам использовать одинаковые инструменты для автоматизации рабочего потока — например, интегрируя инструменты документации в пайплайн развертывания и используя для документации ту же IDE, что и для кода. И это может положительно отразиться на затратах, так как многие инструменты с открытым исходным кодом, которые могут использоваться подобным образом (например, недавно упомянутые статические анализаторы кода), являются бесплатными.

Большое преимущество рассмотрения технической документации как кода заключается в том, что это помогает согласовать рабочие процессы команд разработки и технических писателей, формируя культуру совместного владения документацией. Разработчики обычно пишут черновой вариант документации, который затем улучшают технические писатели. По возможности храните документацию в одном

https://liveinternet.club

репозитории с кодом приложения (например, в папке `docs`). Это упростит обновление документации командой в процессе работы над кодом. Также это означает, что команда может блокировать интеграцию пул-реквеста новой функциональности, пока документация не будет готова. Так как документация хранится в простой текстовой разметке в репозитории исходного кода проекта, над ней смогут работать больше людей в организации.

Эрик Хольшер (Eric Holscher), один из основателей сообщества Write the Docs, поддерживает актуальность вводной страницы концепции Docs as Code по адресу www.writethedocs.org/guide/docs-as-code/.

8.11. Поддержка предложений по дизайну API

В том, что касается этапа исходного кода пайплайна обработки предложений по дизайну API, следует учитывать несколько рекомендаций. Они помогут улучшить пайплайн для пользователей.

8.11.1. Понимание пользователей и их требований

С точки зрения дизайн-ревью API двумя основными группами пользователей пайплайна являются проектировщики API и команда управления API. Понаблюдайте, как они используют пайплайн и с какими проблемами сталкиваются. Проводите встречи по дизайн-ревью API (см. главу 5) для выявления проблем, которые встречаются регулярно, но еще не были автоматизированы. Сохраните полученную информацию и включите ее в список минимальных критериев дизайн-ревью, которые должны проверяться в пайплайне.

Необходимо оптимизировать отчет сборки MDRC для проектировщиков API и команды управления MDRC. А именно при запуске задания формировать отчет, который можно легко прочитать. Он должен включать следующие данные:

- какие MDRC были запущены;
- какие правила статического анализа были проигнорированы;
- какие проблемы статического анализа были обнаружены, но не вызвали сбоя сборки.

На мой взгляд, команда управления API должна выступать полноправным пользователем отчетов пайплайнов, которые должны быть оптимизированы для нее.

8.11.2. Проектирование пайплайна для быстрой обратной связи

Скорость при выполнении пайплайна очень важна. Проектировщики и ревьюеры API хотят, чтобы сборка пул-реквеста обеспечивала быструю обратную

https://liveinternet.club

связь, поэтому автоматизированные проверки на этой стадии должны быть оптимизированы для быстрого выполнения. Как правило, сборка пул-реквеста должна занимать не более 5 минут. Рассмотрите возможность параллельного выполнения разных заданий (как в примере для GitHub Actions) для ускорения сборки.

8.11.3. Простое определение причин сбоя сборки

Если сборка завершается сбоем, проектировщики API должны видеть полезный отчет, отсылающий их к причине сбоя. Причина должна быть видна в пользовательском интерфейсе сервера CI/CD без долгого просмотра журналов. Сообщения об ошибках статического анализа должны содержать достаточно контекста о тех частях файла определения API, в которых были выявлены проблемы, и могут включать ссылки на разделы руководства API, предоставляющие дополнительную информацию о правиле, которое реализуется правилом статического анализа. В таком случае пользователь может перейти по ссылке из лога в раздел стилевого руководства, где объясняется суть проблемы.

Кроме того, при неудачном выполнении пайплайна пул-реквест должен блокироваться, чтобы избежать слияния. Так проектировщик API поймет, что существует проблема, которую надо решить.

8.11.4. Предоставление ссылок на руководство по стилю API при сбоях линтинга

Минимальные критерии дизайн-ревью должны определяться командой управления API, и, если эти критерии не выполняются, сборка должна завершаться неудачей. Обсудите с командой и сверьтесь с руководством по стилю API, чтобы определить, какой уровень проблем линтинга или других обнаруженных проблем должен приводить к сбою пайплайна.

Проверки линтинга должны базироваться на рекомендациях, заданных в руководстве по стилю API. В идеале руководство по стилю должно четко указывать, какие рекомендации обязательны, а какие только желательны; на основе этой информации можно назначать уровни проблем в правиле линтинга (например, ERROR, WARN или HINT).

Использование ключевых слов в стиле RFC 2119 — таких, как MUST («должен»), MUST NOT («не должен») и SHOULD («рекомендуется» или «следует»), — в руководстве по стилю API помогает определить, какой уровень проблемы должен применяться к правилу линтинга. Хороший пример руководства по стилю API, в котором используется RFC 2119, — «Zalando RESTful API and Event Guidelines», доступное по адресу https://opensource.zalando.com/restful-api-guidelines.

https://liveinternet.club

8.11.5. Сбор метрик времени в ходе дизайн-ревью

Промежуток времени от создания пул-реквеста до его слияния называется *временем исполнения* дизайн-ревью. Для организаций, которые тратят слишком много времени на ревью, это важная метрика, которую следует сохранять и отслеживать. Некоторые сервисы размещения кода предоставляют плагины, способные отслеживать и визуализировать эти данные. Если такая возможность недоступна, придется написать скрипты для сохранения и представления этой информации. По мере оптимизации процесса эта метрика должна снижаться.

8.11.6. Включение статических проверок безопасности в сборку пул-реквеста

Безопасность API направлена на защиту информации, передаваемую API, и сокращение рисков, создаваемых уязвимостями API. Безопасность API — большая тема, охватывающая многие части жизненного цикла продукта API, которая будет более подробно рассмотрена в главе 10. Один из вопросов безопасности API, который мне хотелось бы рассмотреть сейчас, — необходимость проверки уязвимостей во время проектирования API. Иногда она называется *статической проверкой безопасности API*. Такие проверки гарантируют, что уязвимости при проектировании API будут обнаружены и исправлены на ранней стадии. Обычно они включают выполнение с файлами определений API некой разновидности линтинга API с правилами, ориентированными на безопасность. Вы можете встроить эти инструменты в сборку пул-реквеста как часть MDRC, чтобы помочь проектировщикам обеспечивать безопасность API уже на ранней стадии.

Многие разработчики средств безопасности API предоставляют коммерческие платформы, которые можно интегрировать в различные части жизненного цикла продукта API для его защиты. Среди таких разработчиков можно выделить 42Crunch (https://42crunch.com), Noname Security (https://nonamesecurity.com), Salt Security (https://salt.security) и Traceable (www.traceable.ai). Вы можете интегрировать их платформы для сканирования уязвимостей в определения API как часть пайплайна развертывания. Эти инструменты также проверяют, что вы следуете лучшим практикам аутентификации, авторизации, передачи и формирования полезных нагрузок данных. Например, 42Crunch предоставляет действие GitHub (https://mng.bz/670A), которое можно включить в пайплайн на базе GitHub Actions. Оно назначает каждому определению API оценку аудита по шкале от 0 (для небезопасных API) до 100 (проблемы не обнаружены). Можно установить минимальную оценку, которую должны набрать определения API, чтобы сборка считалась успешной. При каждом запуске действия формируется ссылка на подробный отчет о безопасности, который можно просмотреть на платформе 42Crunch. Подобные платформы обеспечивают проверки качества безопасности пайплайна.

https://liveinternet.club

8.12. Визуализация определения API для ревью

Вернемся к примеру, над которым мы работали. Реализовав средства контроля, команда управления API в Acme гарантирует, что предложения API, которые она получит на ревью, будут соответствовать минимальным критериям дизайн-ревью. Однако команда управления сталкивается с новой проблемой. Ранее, когда файлы определения API отправлялись им в системе передачи сообщений Slack, команда загружала и открывала их в текстовом редакторе VS Code и преобразовывала в документацию API (с помощью макета представления Swagger UI плагина VS Code OpenAPI). С файлами определений API в пул-реквесте им приходится извлекать ветку пул-реквеста и загружать файлы в свой редактор VS Code. В принципе это нормально, но нет ли более простого способа просмотра и рендеринга файлов определений API в виде документации API, позволяющего обойтись без обязательного извлечения ветки пул-реквеста?

Рендеринг и определение API на этапах проектирования и ревью дизайна — необходимые задачи. Команда Acme может решить эту проблему несколькими способами, которые мы подробно рассмотрим в следующих подразделах.

- Интеграция репозитория исходного кода с контролем версии, содержащего файлы определений API, с инструментами совместной работы над API (ADC, API Design Collaboration), что позволяет просматривать документацию API в инструментах ADC.
- Публикация файла определения OpenAPI на сервере в ходе сборки пул-реквеста и просмотр документации API там.
- Использование плагина репозитория исходного кода, который преобразует файл определения OpenAPI в визуальную документацию API.

8.12.1. Интеграция репозитория исходного кода с ADC

Веб-инструменты ADC — такие, как Stoplight Studio (https://stoplight.io), Apicurio Studio (www.apicur.io/studio), SwaggerHub (https://swagger.io/tools/swaggerhub) и Postman (www.postman.com), — предоставляют пользовательский интерфейс для совместного определения и редактирования файлов определений OpenAPI. Эти инструменты включают множество полезных функций для работы с файлами определений API. Они особенно удобны для проектирования и ревью файлов определений API, поскольку позволяют проектировщикам редактировать источник определения API, переключаться между представлением исходного определения API и представлением документации API, а также добавлять комментарии в файлы (что важно в процессе ревью).

При выборе средств ADC, используемых в рабочем потоке, необходимо учитывать один важный момент: они должны поддерживать систему контроля версий (VCS, Version Control System), упоминавшуюся ранее, как основное хранилище

https://liveinternet.club

файлов определений API. Это означает, что ADC должны читать и записывать информацию в VCS.

ПРИМЕЧАНИЕ Многие разработчики традиционно используют Postman для отправки запросов API, и меня иногда спрашивают, есть ли в Postman функциональность проектирования API. Да, есть — Postman поддерживает создание, импортирование и редактирование определений OpenAPI. Это позволяет подключиться к GitVCS для синхронизации и управления изменениями в определениях API. Узнать больше о возможностях Postman по работе с API можно на https://mng.bz/oenp.

Использование ADC в качестве основного хранилища определений API

Мне знакомы случаи, когда вместо VCS в качестве основного хранилища файлов определений API команды использовали инструменты ADC. В таком сценарии все правки дизайна и ревью файлов определений API выполняются в ADC. Подобное решение может быть привлекательным, потому что в некоторых ADC предусмотрена функциональность реестра API. Иначе говоря, они не только работают как хранилище файлов определений API, но и предоставляют API для поиска, чтения и обновления определений API (реестры API будут подробнее рассмотрены в главе 9). В такой схеме ADC становится исходным репозиторием для файлов определений API. К преимуществам использования ADC в качестве репозитория для файлов определений API можно отнести то, что это самый простой вариант начала работы, особенно для команд, не имеющих опыта использования ADC. Впрочем, у него есть и недостатки.

Во-первых, хотя некоторые ADC поддерживают ограниченную функциональность истории версий, поиска различий в файлах и параллельного редактирования, эти возможности далеки от аналогичной функциональности, предоставляемой специализированными VCS. VCS — идеальные инструменты для хранения истории версий исходных файлов, и это очень важное требование для аудита в любом программном проекте. Одна из фундаментальных практик DevOps требует, чтобы все исходные и конфигурационные файлы проекта хранились в системе контроля версий. В своей книге «Accelerate»[1] Николь Форсгрен (Nicole Forsgren), Джез Хамбл (Jez Humble) и Джин Ким (Gene Kim) (2018, IT Revolution Press) в ходе исследований обнаружили, что время доставки программного продукта сильно коррелирует с контролем версии и автоматизированным тестированием. Они также обнаружили, что частота развертывания сильно коррелирует с применением CD и полнофункциональным контролем версий. Использование упрощенных процессов дизайн-ревью API на основе пул-реквестов, поддерживаемых файлами определений API, позволяет сократить время исполнения для проектирования и доставки. По этой причине второй принцип APIOps, рассматриваемый в главе 1, раздел 1.5.2, требует хранения файлов определений API в системе контроля версий.

[1] Форсгрен Н., Хамбл Д., Ким Д. «Ускоряйся! Наука DevOps. Как создавать и масштабировать высокопроизводительные цифровые организации».

https://liveinternet.club

Во-вторых, использование ADC как основного хранилища для файлов определения API может ограничить возможности интеграции. Такие VCS, как Git, и такие сервисы размещения кода, как GitHub, GitLab и Bitbucket, предлагают отличные возможности интеграции с серверами сборки, системами ведения тикетов, IDE, системами качества кода и т. д. Таким образом расширяются возможности интеграции для исходных файлов определений API, хранящихся в VCS.

И в-третьих, использование ADC как основного хранилища для файлов определения API может привязать данные (файлы с исходным кодом) к инструментам их редактирования (ADC). Это может создать трудности для разработчиков и других пользователей, которые предпочитают другие инструменты для редактирования.

8.12.2. Публикация файла определений OpenAPI для пул-реквеста на сервере

Файлы определений OpenAPI можно публиковать на сервере, который преобразует их. Сервер может визуализировать файлы с использованием SwaggerUI, Redocly, Stoplight Elements и других стилей. Полезный инструмент с открытым исходным кодом для этой задачи — Swagger UI Watcher (https://github.com/moon0326/swagger-ui-watcher). Чтобы сделать рабочий поток разработчика более динамичным, инструмент обнаруживает изменения в файле определений OpenAPI в файловой системе и перезагружает его в Swagger UI в браузере.

8.12.3. Использование плагина для просмотра исходного репозитория

Некоторые сервисы размещения кода поддерживают рендеринг файлов определений OpenAPI прямо в репозитории исходного кода (или имеют плагины, обеспечивающие такую поддержку). Это означает, что при проверке пул-реквеста ревьюеры могут просмотреть его и увидеть файл определения API. В качестве примера можно привести программу просмотра Bitbucket OpenAPI (https://mng.bz/aEpm). Также существуют плагины для браузера Chrome, выполняющие рендеринг файлов определений OpenAPI; вы найдете их в интернет-магазине Chrome.

8.13. Повторное использование объектов API: мультифайловые определения API

В ходе работы над крупными программными проектами, где участвуют несколько команд, могут создаваться монолитные файлы OpenAPI, иногда насчитывающие десятки тысяч строк. При неправильном управлении такими файлами в них могут появиться дублирующиеся объекты OpenAPI, их обновление станет слишком утомительным, а повторное использование объектов усложнится.

https://liveinternet.club

Если редактированием этих файлов занимаются несколько команд, растет риск файловых конфликтов в системе контроля версий. Большой размер также усложняет ревью файлов, а в некоторых инструментах OpenAPI (редакторах, API-шлюзах и т. д.) могут возникнуть трудности с их открытием. Как же управлять большими файлами определений API?

С точки зрения дизайна API команды могут предотвратить чрезмерное разрастание файлов определений, четко определив границы API и их область действия (подробнее см. в книге Джеймса Хиггинботэма (James Higginbotham) об определении границ API с правильной детализацией «Principles of Web API Design» (Addison-Wesley Professional, 2021)). Но с точки зрения файла определения API определение должно содержать ссылки на повторно используемые объекты (или такие ссылки могут появиться в результате рефакторинга). Повторно используемые объекты, на которые указывают эти ссылки, должны находиться в разделе `Components Object` определения. Согласно OAS 3.1, `Components Object` содержит повторно используемые объекты: схемы, ответы, параметры, примеры, тела запросов, заголовки, схемы безопасности, ссылки, обратные вызовы и элементы пути.

Для создания ссылок используются объекты ссылок OpenAPI. Объект ссылки содержит поле `$ref` с URI-идентификатором, определяющим целевой объект, на который указывает ссылка. Для внутренних ссылок (то есть ссылок в границах того же документа) URI ссылки может состоять только из решетки (`#`), за которой следует идентификатор фрагмента. Пример:

```
/v1/catalog/products/{id}/reviews:
  get:
    tags:
      - Reviews
    summary: Список обзоров
    description: Список всех обзоров продукта.
    operationId: listProductReviews
    parameters:
      - $ref: '#/components/parameters/productId'
    responses:
      '200':
        description: OK
        headers:
          RateLimit-Limit:
            $ref: '#/components/headers/rateLimit'
          RateLimit-Remaining:
            $ref: '#/components/headers/rateLimitRemaining'
          RateLimit-Reset:
            $ref: '#/components/headers/rateLimitReset'
        content:
          application/vnd.api+json:
            schema:
              $ref: '#/components/schemas/ReviewsResponse'
```

https://liveinternet.club

Использование объектов ссылок для внутренних компонентов помогает, но полученный файл определения API все еще достаточно велик. Чтобы оптимизировать размер файла, объекты можно переместить из файла определения API во внешние фрагменты или дочерние документы. Полученный *мультифайловый документ определения API* состоит из родительского документа и ссылок на дочерние документы. URI в ссылке обозначает абсолютный или относительный путь к документу, за которым следует указатель JSON, определяющий объект в документе (рис. 8.9).



Рис. 8.9. Структура внешней ссылки OpenAPI

8.13.1. Разбиение определения OpenAPI

Каталог `chapter8/apis/product-catalog-api` содержит пример файла OpenAPI, который был разбит по следующим правилам.

- Каждое родительское определение API должно иметь собственную папку. Эта папка содержит две вложенные папки — `paths` и `components`. Папка `components` содержит вложенную папку для каждого типа компонентов, входящих в API (например, заголовки, параметры, схемы, ответы, примеры и т. д.).
- Каждый объект пути в родительском определении API должен ссылаться на объект пути, находящийся в отдельном файле в каталоге пути. Этот файл может содержать несколько операций (например, GET, POST, PUT и т. д.).
- Каждый элемент заголовка должен иметь отдельный файл в каталоге `headers`.
- Каждая схема успешного (2xx) ответа должна иметь отдельный файл в каталоге `responses`.
- Каждый объект ответа об ошибке должен иметь отдельный файл в каталоге `responses`.
- Каждый файл должен быть допустимым документом OpenAPI.
- Все ссылки должны содержать обе части: файл и указатель JSON.

Итоговая структура каталогов показана на рис. 8.10.

https://liveinternet.club

```
→  product-catalog-api git:(workbranch) tree .
.
├── components
│   ├── headers
│   │   ├── rateLimit.yaml
│   │   ├── rateLimitRemaining.yaml
│   │   └── rateLimitReset.yaml
│   ├── parameters
│   │   ├── filterCategory.yaml
│   │   ├── include.yaml
│   │   └── productId.yaml
│   ├── responses
│   │   ├── 400Response.yaml
│   │   ├── 401Response.yaml
│   │   ├── 403Response.yaml
│   │   ├── 404Response.yaml
│   │   ├── 406Response.yaml
│   │   ├── 415Response.yaml
│   │   ├── 429Response.yaml
│   │   └── 500Response.yaml
│   ├── schemas
│   │   ├── CategoriesResponse.yaml
│   │   ├── Category.yaml
│   │   ├── Error.yaml
│   │   ├── Link.yaml
│   │   ├── Pagination.yaml
│   │   ├── Product.yaml
│   │   ├── ProductsResponse.yaml
│   │   ├── Review.yaml
│   │   └── ReviewsResponse.yaml
│   └── securitySchemes
│       └── OAuth2.yaml
├── openapi.yaml
└── paths
    ├── v1-catalog-categories.yaml
    ├── v1-catalog-products-id-reviews.yaml
    ├── v1-catalog-products-id.yaml
    └── v1-catalog-products.yaml
```

Рис. 8.10. Пример структуры каталогов с мультифайловым документом OpenAPI

Вот как будут выглядеть объекты путей и компонентов в файле определения родительского API с использованием указанных правил:

```
openapi: 3.0.3
info:
  title: API каталога продуктов
        version: 0.0.1
…
tags:
  - name: Categories
    description: Категория – это класс продуктов
➥ with common characteristics
…
paths:
  /v1/catalog/categories:
    $ref: 'paths/v1-catalog-categories.yaml#
➥ /paths/~1v1~1catalog~1categories'
```

Ссылка включает файл и указатель JSON на целевой объект

https://liveinternet.club

```
  /v1/catalog/products:
    $ref: 'paths/v1-catalog-products.yaml#
➥ /paths/~1v1~1catalog~1products'
  /v1/catalog/products/{id}:
    $ref: 'paths/v1-catalog-products-id.yaml#
➥ /paths/~1v1~1catalog~1products~1{id}'
  /v1/catalog/products/{id}/reviews:
    $ref: 'paths/v1-catalog-products-id-reviews.yaml#
➥ /paths/~1v1~1catalog~1products~1{id}~1reviews'
components:
  securitySchemes:
    $ref: 'components/securitySchemes/OAuth2.yaml#
➥ /components/securitySchemes'
security:
  - OAuth2: []
```

Каждый объект пути хранится в отдельном файле

Объект пути `v1-catalog-categories.yaml`, на который указывает ссылка, выглядит так:

```
openapi: 3.0.3
info:
  title: API каталога продуктов Acme
  version: 0.0.1
paths:
  /v1/catalog/categories:
   get:
     tags:
       - Categories
     summary: Список всех категорий
     description: Список всех категорий
     operationId: listCategories
     parameters: []
     responses:
       '200':
         description: OK
         headers:
           RateLimit-Limit:
             $ref: '../components/headers/rateLimit.yaml#
➥ /components/headers/rateLimit'
           RateLimit-Remaining:
             $ref: '../components/headers/rateLimitRemaining.yaml#
➥ /components/headers/rateLimitRemaining'
           RateLimit-Reset:
             $ref: '../components/headers/rateLimitReset.yaml#
➥ /components/headers/rateLimitReset'
         content:
           application/vnd.api+json:
             schema:
               $ref: '../components/schemas/CategoriesResponse.yaml#
➥ /components/schemas/CategoriesResponse'
       '401':
         $ref: '../components/responses/401Response.yaml#
➥ /components/responses/401Response'
       '403':
         $ref: '../components/responses/403Response.yaml#
```

Это файл OpenAPI

Ссылка на объект заголовка из файла в каталоге headers

Ссылка на объект ответа из файла в каталоге responses

https://liveinternet.club

```
➥ /components/responses/403Response'
      '406':
        $ref: '../components/responses/406Response.yaml#
➥ /components/responses/406Response'
….
```

OAS не устанавливает никаких правил относительно структуры дочерних документов JSON/YAML, на которые ссылается корневой документ. Тем не менее структурирование дочерних документов как полностью действительных документов OAS означает, что к ним может применяться линтинг и они могут преобразовываться и обновляться до другой версии OAS с использованием OAS-совместимых инструментов. Это также означает, что с ними можно работать в редакторах, поддерживающих OpenAPI, и пользоваться всей помощью, которую они предоставляют. Поле `$.info.version` в дочернем документе позволяет при необходимости версионировать фрагмент отдельно от корневого документа.

ПРИМЕЧАНИЕ Майк Ральфсон (Mike Ralphson), редактор OAS, подробно рассказывает о создании определений OpenAPI дочерних документов на своем веб-сайте: https://mng.bz/Y74z.

8.13.2. Объединение мультифайловых документов

Чтобы объединить мультифайловые определения API в один файл (например, для использования в инструментах, не поддерживающих внешние ссылки), можно выполнить их *сборку* (bundling). В процессе сборки необходимо извлечь внешние ссылки из разных файлов или URL и добавить их в компонентный объект определения OpenAPI. Это можно сделать командой `bundle` Redocly CLI или Swagger CLI: https://github.com/APIDevTools/swagger-cli (их функции сборки схожи). В своем примере проекта опробуйте эту возможность, выполнив сборку определения в папке `product-catalog-api`:

```
cd apis/product-catalog-api
redocly bundle openapi.yaml --output bundled.yaml
```

Просмотрите созданный файл `bundled.yaml` и убедитесь, что все внешние ссылки консолидированы в виде внутренних ссылок.

СОВЕТ Я уже упоминал средства командной строки для объединения файлов определений API. Но если вам нужна встраиваемая библиотека JavaScript, которая может использоваться для сборки определений, рассмотрите JSON Schema $Ref Parser: https://github.com/APIDevTools/json-schema-ref-parser.

8.13.3. Что необходимо учитывать при работе с мультифайловыми определениями API

Мультифайловые документы определения API помогают сделать определения более модульными и простыми для понимания и обслуживания. Они также

https://liveinternet.club

стимулируют повторное использование и инкапсуляцию, так как переиспользуемые компоненты в отдельных файлах могут быть общими для нескольких родительских определений API. Таким образом устраняется дублирование и обеспечивается согласованность в документации API. Мультифайловые определения API также упрощают работу нескольких команд над разными частями документа и снижают риск конфликтов слияния.

Тем не менее мультифайловые определения API создают определенные сложности. Чтобы избежать путаницы, необходимо выполнять операции с файлами, используя понятные соглашения об именах. Кроме того, разработчику придется выделить время на изучение структуры файлов и отношений между компонентами. OAS поддерживает мультифайловые документы, но этого нельзя сказать обо всех инструментах и фреймворках. Очень важно проверить, что используемые вами инструменты совместимы с мультифайловыми документами, чтобы избежать любых непредвиденных ограничений.

8.13.4. Инструменты для работы с мультифайловыми определениями API

Компоненты, пригодные для многократного использования, можно вручную создавать в файлах определений API. Однако работа с YAML- и JSON-файлами в текстовом редакторе однообразна и утомительна. Такие средства совместного проектирования API, как Stoplight, Apicurio и SwaggerHub, поддерживают повторно используемые компоненты OpenAPI и могут упростить задачу. Они позволяют создать библиотеку переиспользуемых компонентов API, на которые вы можете ссылаться в своих файлах определений.

Но представьте, что у вас имеется один большой файл определения OpenAPI, который вы хотите разделить и преобразовать в мультифайловое определение. Это может быть одноразовая задача или действие, которое вам хотелось бы выполнять в своем пайплайне развертывания. Для этого можно воспользоваться функцией Redocly CLI `split`. Опробуйте эту возможность из каталога `chapter8/apis`:

```
redocly split product-catalog.oas.yaml --outDir out
```

Преимущество использования Redocly CLI для разделения файлов состоит в том, что система поддерживает настраиваемые соглашения об именовании полученных файлов. Обратите внимание: полученные дочерние документы содержат только ссылочные объекты и сохраняются в формате YAML или JSON, не являясь стандартными документами OpenAPI. Они могут быть полезными и в этой форме, но чтобы они стали действительными документами OpenAPI, их потребуется обновить.

https://liveinternet.club

Итоги

- Команды управления API могут определять минимальные критерии дизайн-ревью (MDRC) — набор автоматизированных проверок, которые должно пройти предложение API, прежде чем передавать соответствующий пул-реквест на ревью. Эти автоматизированные проверки в пайплайне должны включать линтинг API, проверки критичных изменений и любые другие автоматизированные проверки, которые команда управления сочтет важными.
- Шаблоны пул-реквестов напоминают проектировщикам API о том, что следует предоставить контекст: почему рассматриваемое изменение API необходимо. Это поможет рецензентам пул-реквестов понять причины и область действия изменений и ускорит процесс ревью.
- Назначение владельцев кода для файлов — механизм, требующий, чтобы эксперты предметной области проводили ревью и одобряли изменения файлов в исходном репозитории команды. Вы можете назначить участников команды управления API владельцами кода для файлов определений API, чтобы все изменения гарантированно поступали им на ревью.
- Статические анализаторы (линтеры) текстов — такие, как Vale, — могут выполняться в сборке пул-реквеста для проверки файлов руководства API на правильность пунктуации, орфографии, грамматики и стиля, а также на удобочитаемость.
- Инструменты совместной работы при проектировании API позволяют интегрироваться с исходным репозиторием, содержащим файлы определений API. Проследите, чтобы исходный репозиторий с контролем версий был основным хранилищем файлов определений API.
- Разбиение определений API на несколько файлов делает их более управляемыми и повышает качество совместной работы над ними. Оно также облегчает повторное использование общих компонентов API.

https://liveinternet.club

9 CI/CD для артефактов API. 2: стадия сборки и развертывание конфигурации API

В этой главе

- ✓ Преобразование определений API в пайплайне
- ✓ Публикация сгенерированных артефактов API в хранилище артефактов
- ✓ Создание SDK и мок-API
- ✓ Развертывание конфигурации API-шлюза

В главе 8 я описывал средства управления на этапе исходного кода, которые помогают командам применять автоматизированные проверки к определениям API на стадии проектирования. После ревью и одобрения определений команда API может:

- сгенерировать артефакты из определения API. Это могут быть SDK, альтернативные форматы определений API и документация API для портала разработчика в формате, понятном для человека;
- развернуть сервис API в промежуточных средах;
- развернуть конфигурацию API-шлюза;
- провести тестирование реализованного API на соответствие.

https://liveinternet.club

В этой главе рассматривается выполнение этих задач на двух этапах: сборки, а также развертывания и тестирования (рис. 9.1).

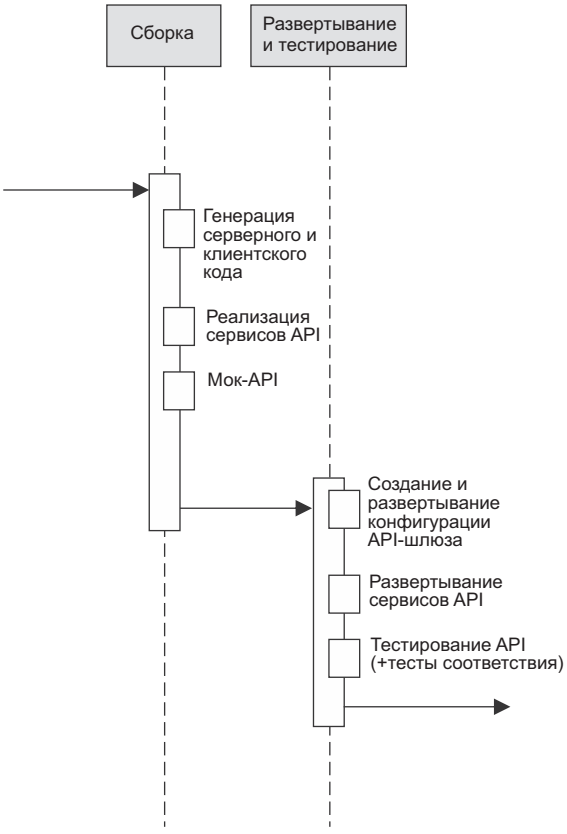


Рис. 9.1. Практики APIOps, рассматриваемые в этой главе

Эта практическая глава предназначена в основном для разработчиков, которым требуется создать пайплайн для развертывания и публикации API. Мы разберем пример создания пайплайна для генерации и публикации артефактов API во внутренних хранилищах артефактов с использованием GitHub Actions. Развертывание конфигурации API будет реализовано в двух API-шлюзах с открытым исходным кодом с использованием рабочего потока APIOps (то есть на базе GitOps).

Хотя эта глава в первую очередь предназначена для разработчиков, высокоуровневое представление основных стадий пайплайна для артефактов API также пригодится владельцам продуктов API. Им будет полезно понимать, какие команды участвуют в рабочем потоке развертывания APIOps и как он

https://liveinternet.club

вписывается в топологию команд организации. Владельцам продуктов API и менеджерам продуктов может пригодиться информация об управлении SDK и затруднениях, возникающих при их создании.

> **ПРИМЕЧАНИЕ** Примеры, приведенные в этой главе, продолжают и расширяют код главы 8, который можно найти на странице https://mng.bz/ngo2. В главе вы найдете также дополнительный код: https://mng.bz/GZAD и https://mng.bz/z8O1. Я рекомендую воспроизвести эти примеры, но на случай, если вы предпочтете сразу перейти к готовому решению, я создал в репозитории соответствующие файлы с суффиксом `.completed` (там, где это возможно). Чтобы эти файлы работали как задумано, их необходимо переименовать.

Опыт работы с Docker и Kubernetes желателен, но не обязателен. Краткое введение в эти технологии приводится в приложении Д.

9.1. Этап сборки: преобразование файлов определений API

Представьте, что перед публикацией файлов определений API на портале разработчика необходимо сделать следующее.

1. Собрать мультифайловое определение в один файл.
2. Отфильтровать полученный файл определения. Допустим, удалить все объекты OpenAPI, помеченные расширением OpenAPI `x-internal:true`.
3. Отсортировать поля `operationId` в файле определения OpenAPI, чтобы они следовали в определенном порядке. Например, сначала поле `operationId`, затем `summary` и далее `description`, `parameters`, `requestBody` и `responses` именно в таком порядке.
4. Удалить из файла определения любые неиспользованные компоненты.
5. Создать коллекцию Postman из файла OpenAPI.
6. Отправить обработанный файл определения OpenAPI в хранилище артефактов API, чтобы на более поздних этапах пайплайна его можно было загрузить, использовать или опубликовать на портале разработчика.

> **ПРИМЕЧАНИЕ** В главе 4 я представил расширения OpenAPI как механизм включения дополнительных данных в файлы определений API. Например, это может быть полезно в ситуациях, когда файл определения содержит эндпоинты как для внутреннего, так и для внешнего использования. Объекты внутренних эндпоинтов можно пометить расширением (например, `x-internal:true`), чтобы отличать их от внешних эндпоинтов. Использование расширения в файле определения OpenAPI означает, что инструменты, понимающие это расширение, смогут обрабатывать его нужным образом, а другие инструменты могут его просто игнорировать.

https://liveinternet.club

Я называю эти изменения в файле определения API *преобразованиями* (transformations). Какое бы конкретное преобразование вы ни выполняли, оно должно автоматизироваться в пайплайне развертывания, чтобы определение API подходило для внешней публикации или конкретного пользователя файла определения. Внешнее представление может создаваться в результате фильтрации, переименования, перезаписи, сборки, слияния нескольких определений API или других преобразований. Преобразование необходимо, когда организация поддерживает свое представление файла определения API, отличное от опубликованного для внешних потребителей. Другой типичный пример — необходимость замены URL сервера, указанного в определении API, с внутреннего сервера на внешний сервер-песочницу.

> **ПРИМЕЧАНИЕ** Под термином *сборка* (bundling) в этом контексте я подразумеваю объединение мультифайлового определения API в один файл определения, а под термином *слияние* (merging) — объединение двух и более автономных определений API в один файл.

Преобразованные определения API можно поместить в *хранилище артефактов* API, откуда они извлекаются командами разработки и приложениями. Конкретные формы реализации хранилища артефактов API могут быть разными: *реестр API*, *рабочее пространство Postman* и даже хранилище двоичных артефактов (такое, как JFRog Artifactory или GitHub Packages). Реестр API хранит файлы определений API и предоставляет API, при помощи которого пользователи могут отправлять, искать и читать их. Рабочее пространство Postman напоминает каталог на платформе Postman для работы и совместного использования определений API, сохраненных коллекций запросов API и файлов среды запросов API. Какую бы разновидность хранилища вы ни выбрали, важно, чтобы после генерации артефактов на этапе непрерывной интеграции (CI) пайплайна развертывания последующие этапы пайплайна могли извлечь артефакты для использования.

На рис. 9.2 показано, как на этапе CI происходит чтение мультифайлового определения API из репозитория исходного кода, его преобразование и сохранение в хранилище артефактов.

Чтобы понять, как будет использоваться артефакт API — в данном случае обработанный файл определения API, — вы можете составить сводку пользователей артефакта и инструментов, используемых для генерации артефакта, как, например, в табл. 9.1. Ведение списка артефактов API, генерируемых пайплайном, в такой таблице поможет следить за тем, какие артефакты вы генерируете, кто является пользователями и для чего им нужны артефакты. Это помогает сосредоточиться на потребностях пользователей артефакта.

Я рассказал о преобразованиях определений API и сохранении их в хранилище артефактов API как части CI-этапа пайплайна. Теперь напишем код для выполнения этих операций в пайплайне.

https://liveinternet.club

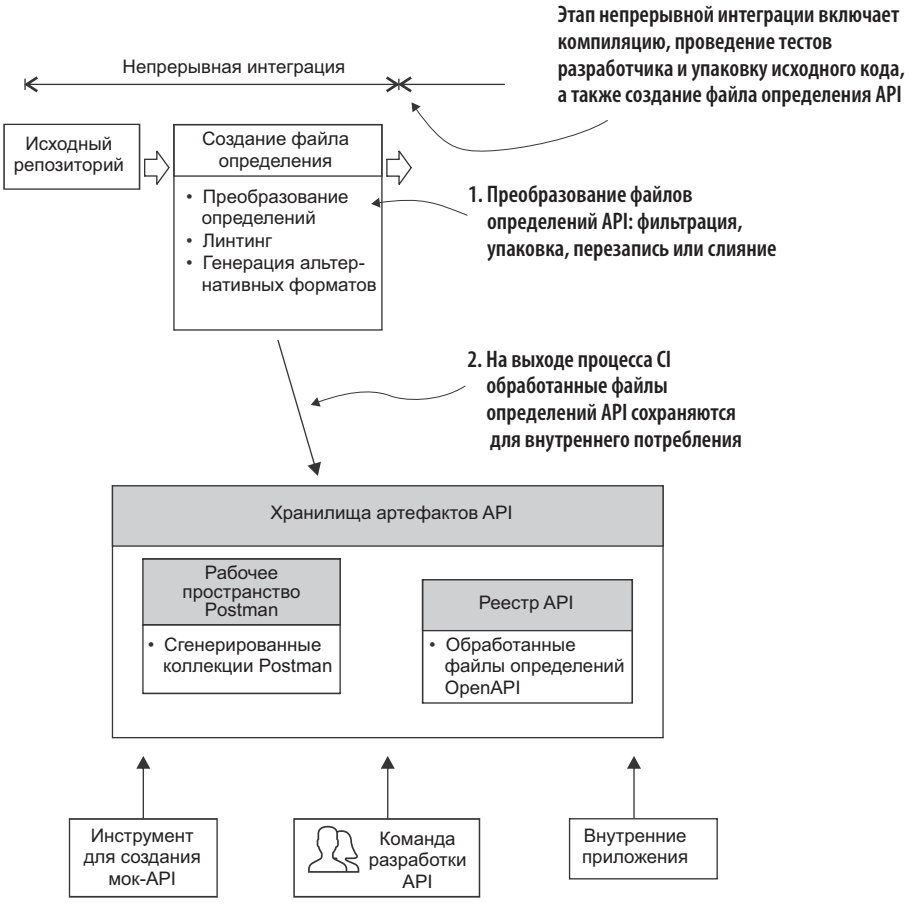


Рис. 9.2. Пример преобразования определения API в пайплайне

9.1.1. Обновление кода пайплайна

Как правило, CI-этап пайплайна развертывания включает извлечение исходного кода, компиляцию, проведение быстрых тестов разработчика (также называемых юнит-тестами), упаковку откомпилированного кода в исполняемый файл и сохранение последнего в хранилище двоичных артефактов. Кроме этих задач пайплайн должен выполнять преобразования с файлами определений API, а затем сохранять файл определения в хранилище артефактов API. Следующий пример показывает, как это сделать.

Для начала создайте новую ветку для ваших изменений на основе главной ветки. Создайте файл рабочего потока GitHub, который будет строить приложение при слиянии пул-реквеста. Создайте в репозитории новый файл рабочего потока `.github/workflows/build.yaml`, как показано в листинге 9.1.

https://liveinternet.club

Таблица 9.1. Сводка генерируемых артефактов определений API

Тип артефакта	Пользователи артефакта	Для чего им нужен артефакт	Исходные файлы, на основе которых генерируется артефакт	Инструмент для генерации артефакта	Где будет храниться артефакт
Обработанное определение OpenAPI	• Внешние потребители API • Портал разработчика • Более поздние этапы пайплайна непрерывной интеграции / непрерывного развертывания (CI/CD)	• Совместно используется внешними потребителями API • Используется для генерации ссылки на API на портале разработчика • Используется для генерации SDK	Мультифайловые определения API	Redocly CLI, openapi-format, openapi-filter	Реестр API Apicurio

Листинг 9.1. Создание файла рабочего потока GitHub

```
name: Deployment pipeline

on:
  push:
    branches:
      - main
env:
  GITHUB_RUN_NUMBER: ${{ github.run_number }}
```

В дополнение к этому файлу создайте задание с именем `pipeline`, которое будет извлекать исходный код приложения и компилировать его в исполняемый файл `.jar`. Задание должно иметь разрешения на чтение содержимого репозитория и запись пакетов (в GitHub Packages https://github.com/features/packages — репозиторий двоичных артефактов GitHub для публикации и потребления программных пакетов). В листинге 9.2 показано, как это делается.

Листинг 9.2. Создание задания сборки

```
jobs:
  pipeline:
    name: Pipeline
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
```

https://liveinternet.club

```
    steps:
      - name: Build - Checkout
        uses: actions/checkout@v3
      - name: Build - Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven
      - name: Build - Check Java version
        run: |
          mvn -version
          echo $JAVA_HOME
      - name: Build - Create application jar
        run: |
          cd chapter8
          ./run_app_build.sh
```

При создании микросервисов с использованием CI/CD приложения обычно компилируются в исполняемые файлы и упаковываются в контейнеры (например, при помощи технологии Docker, рассматриваемой в приложении Д к книге). Контейнеры содержат все библиотеки и зависимости, необходимые приложениям для выполнения, и хранятся в реестре контейнеров. Далее их можно загружать и выполнять в других последующих средах в пайплайне.

> **ПРИМЕЧАНИЕ** Код приложения для создания сервиса API в этом примере не предназначен для демонстрации лучших практик создания приложений Spring Boot с помощью Maven; в нем я использую несколько ускоренных решений.

Чтобы построить приложение из этого примера, добавьте в задание шаг упаковки созданного исполняемого файла `.jar` в образ Docker и сохранения его в реестре GitHub Container Registry (GHCR), как показано в листинге 9.3.

Листинг 9.3. Сохранение образа контейнера приложения в GHCR

```
      - name: Build - Log in to the GitHub Container Registry
        uses: docker/login-action@65b78e6e13532edd9afa3aa52ac7964289d1a9c1
        with:
          registry: ghcr.io
          username: ${{ github.repository_owner }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - name: Build - Build and push Docker image
        uses: docker/build-push-action
➥ @f2a1d5e99d037542a71f64918e516c093c6f3fc4
        with:
          context: .
          push: true
          tags: ghcr.io/${{ github.repository_owner }}
➥ /products-api-service:${{ github.run_number }}
          file: chapter8/Dockerfile
```

https://liveinternet.club

Теперь напишем шаги для преобразования определений API в пайплайне. Мультифайловое определение API необходимо собрать в один файл, для этого воспользуйтесь командой Redocly CLI `bundle`. Добавьте шаг сборки файла определения API и переместите полученный файл в каталог `build-output`, как показано в листинге 9.4.

Листинг 9.4. Сборка файла определения API

```
- name: Build - Create output directory for API artifacts
  run: mkdir build-output
- name: Build - Bundle API definition file
  run: |
    cd chapter8/apis/product-catalog-api
    npx @redocly/cli bundle openapi.yaml --output api.bundled.yaml
    mv api.bundled.yaml ../../../build-output/api.bundled.yaml
```

Затем результат фильтруется для удаления внутренних эндпоинтов и неиспользуемых компонентов, а поля объектов OpenAPI сортируются в нужном порядке. Примеры инструментов, которые помогут отфильтровать определение OpenAPI — openapi-filter Майка Ральфсона (Mike Ralphson) (https://github.com/Mermade/openapi-filter), thim81/openapi-format (https://github.com/thim81/openapi-format), а также функции фильтров и специализированные плагины Redocly CLI (https://github.com/Redocly/redocly-cli). Вы даже можете написать собственный код, чтобы выполнить парсинг определений API и представить их в виде объекта в памяти, а далее отфильтровать по своему усмотрению.

Для нашего примера напишите шаг для фильтрации всех объектов OpenAPI с расширением `x-internal:true`, к числу которых относится эндпоинт `GET /v1/catalog/products/{id}/reviews`, с использованием инструмента Mermade/openapi-filter. Команды Mermade/openapi-filter выполняются в формате `openapi-filter` <*параметры*> `--` <*входной_файл*> <*выходной_файл*>. Затем создайте шаг для фильтрации всех неиспользуемых компонентов и отсортируйте поля OpenAPI при помощи инструмента `thim81/openapi-format`. Чтобы использовать `thim81/openapi-format`, передайте фильтруемое определение API, а также конфигурации фильтрации и сортировки в формате `openapi-format` <*определение-openapi*> `--filterFile` <*конфигурацияФильтра*> `--sortFile` <*конфигурацияСортировки*>. Листинг 9.5 показывает, как реализовать эти шаги.

Листинг 9.5. Фильтрация файла определения с использованием openapi-format

```
 - name: Build - Filter internal endpoints
   run: |
     npx openapi-filter --flags x-internal --verbose
➥ -- build-output/api.bundled.yaml build-output/api.yaml
- name: Build - Remove unused components
  run: |
```

Использование openapi-filter для фильтрации файла определения

https://liveinternet.club

Удаление неиспользуемых компонентов и сортировка файла определения

```
    npx openapi-format build-output/api.yaml
➥ --filterFile chapter9/unused-components-filter.yaml
➥ --sortFile chapter9/sort.json
➥ --output build-output/api.yaml
- name: Build - Run API linting
  run: |
    npx @stoplight/spectral-cli lint build-output/api.yaml
➥ --ruleset chapter8/apis/rulesets/json-api.ruleset.yaml
```

Файл конфигурации фильтра для `thim81/openapi-format` определяет конфигурацию выполняемой фильтрации — в данном случае виды неиспользуемых компонентов, которые должны быть исключены. Определите конфигурацию в файле `chapter9/unused-components-filter.yaml`, как показано в листинге 9.6.

Листинг 9.6. Файл фильтра openapi-format

```
unusedComponents:
 - schemas
 - parameters
 - examples
 - headers
 - requestBodies
 - responses
```

Я создал для сортировки файл конфигурации `chapter9/sort.json`. Приведенный ниже код задает порядок, в котором поля всех объектов OpenAPI должны упорядочиваться иерархически:

```
{
  "root": ["openapi", "info", "servers", "paths", "components",
➥ "tags", "x-tagGroups", "externalDocs"],
  "get": ["operationId", "summary", "description", "parameters",
➥ "requestBody", "responses"],
  "post": ["operationId", "summary", "description", "parameters",
➥ "requestBody", "responses"],
  "put": ["operationId", "summary", "description", "parameters",
➥ "requestBody", "responses"],
  "patch": ["operationId", "summary", "description", "parameters",
➥ "requestBody", "responses"],
  "delete": ["operationId", "summary", "description", "parameters",
➥ "requestBody", "responses"],
  "parameters": ["name", "in", "description", "required", "schema"],
  "requestBody": ["description", "required", "content"],
  "responses": ["description", "headers", "content", "links"],
  "content": [],
  "components": ["parameters", "schemas"],
  "schema": ["description", "type", "items", "properties", "format",
➥ "example", "default"],
  "schemas": ["description", "type", "items", "properties", "format",
➥ "example", "default"],
  "properties": ["description", "type", "items", "format", "example",
➥ "default", "enum"]
}
```

https://liveinternet.club

Сохраните изменения в репозитории GitHub. Ваш рабочий поток GitHub Actions должен работать успешно.

Итак, вспомним, что же мы сделали. Вы увидели, что существуют разные способы обработки файла определения OpenAPI. В пайплайне можно переименовать файл определения API, преобразовать файл определения из YAML в JSON (или JSON в YAML), удалить неиспользуемые компоненты, отсечь специализированные расширения разработчика и отфильтровать объекты. Полученный файл определения должен где-то храниться, и вы можете сохранить его в реестре API. Реестры API рассматриваются в следующем разделе.

9.1.2. Сохранение определений API в реестре API

Реестр API выполняет в организации роль каталога API. Реестр API — это исполнительная система, которая позволяет пользователям отправлять и публиковать свои файлы определений API и компоненты для других пользователей и программных систем. Реестры API также имеют API, обеспечивающий программный доступ к артефактам API для клиентских приложений. Иначе говоря, они предоставляют API к определениям API организации. Примеры реестров API — Apicurio Registry (www.apicur.io/registry), SwaggerHub (https://mng.bz/0GeJ) и Apigee Registry API (https://github.com/apigee/registry). Клиентские приложения могут использовать определение API или схемы событий для проверки сообщений во время выполнения. Возможность поиска и обнаружения определений API также помогает повысить единообразие и расширить возможности повторного использования схем и компонентов в рамках организации.

Выше я упоминал Apicurio Registry как один из примеров реестров API с открытым исходным кодом. Это часть открытого проекта Apicurio, спонсируемого Red Hat, и она поддерживает создание, обновление и удаление определений API в форматах OpenAPI, AsyncAPI, GraphQL, Apache Avro, Google Protocol Buffers, JSON Schema, Kafka Connect Schema, Web Service Description Language (WSDL) и XML Schema Definitions (XSD).

В Apicurio Registry элементы реестра (такие, как файлы определений OpenAPI и схемы Avro) называются *артефактами*. Коллекция логически связанных артефактов API называется *группой артефактов*. Артефактами в группе обычно управляет одна сущность, приложение или организация. Apicurio Registry также поддерживает использование ссылок на артефакты, где файлы определений API или схемы могут ссылаться друг на друга. Это позволяет многократно использовать схемы и компоненты API. Артефактами можно управлять на программном уровне из Apicurio REST API. На момент написания книги документация API для этой возможности доступна по адресу https://mng.bz/KZrX. А сейчас разберемся, как сохранять и просматривать элементы в Apicurio Registry.

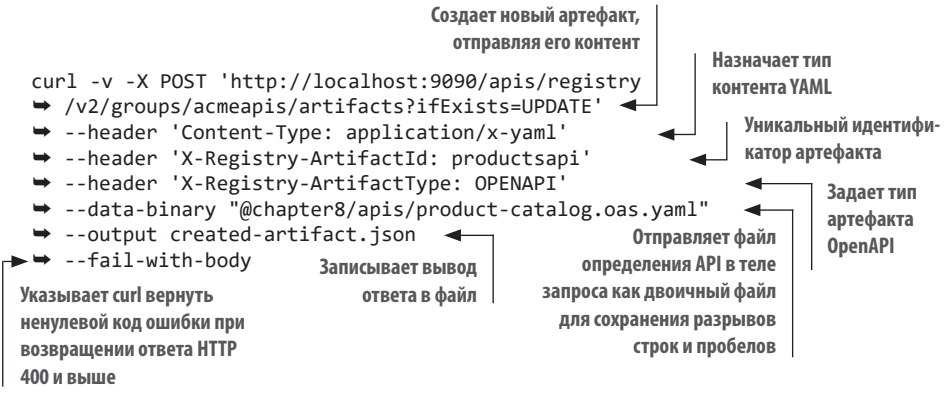
https://liveinternet.club

Публикация в реестре API с компьютера

В корневом каталоге проекта выполните следующую команду, чтобы запустить Apicurio в контейнере `docker` (вместе с API-шлюзом Kong):

```
docker compose -f chapter9/compose.yaml up -d
```

Чтобы увидеть интерфейс Apicurio, перейдите в браузере по адресу http://localhost:9090/ui/artifacts. Теперь отправьте определение API в Apicurio. Не выполняйте операцию через пользовательский интерфейс, а используйте REST API Apicurio. Выполните команду `./chapter9/publish-to-apicurio.sh`, которая выполняет следующий запрос `curl`:



Чтобы просмотреть отправленный файл определения Products API, в браузере перейдите по адресу http://localhost:9090/ui/artifacts/acmeapis/productsapi/versions/latest. Щелкните на раскрывающемся списке Version в правой части страницы, чтобы увидеть опубликованные версии. На вкладке Documentation выводится отрендеренное определение API (в формате ReDoc; https://github.com/Redocly/redoc), а на вкладке Content — необработанное определение API.

Для поиска артефактов API можно воспользоваться средствами поиска в UI. Перейдите на домашнюю страницу http://localhost:9090/ui/artifacts и проведите поиск артефактов со словом catalog в описании. Не забудьте переключить селектор поиска из стандартного положения Name в Description. В реестре присутствует всего один артефакт — Product Catalog API, но он должен появиться в результатах поиска.

Вы увидели, как использовать реестр API локально. Теперь обсудим, как сделать это в пайплайне GitHub Actions.

Публикация в реестре API из пайплайна

В идеале у вас должен быть долгоживущий экземпляр реестра API для публикации, чтобы другие команды и приложения могли впоследствии извлекать из него артефакты. Но для простоты в этом примере вы можете запустить экземпляр

https://liveinternet.club

реестра API в контейнере раннера GitHub Actions (который будет уничтожен по завершении выполнения пайплайна) и публиковать артефакты в нем. Для этого после шага извлечения, созданного ранее, добавьте в файл `.github/workflows/build.yaml` шаг для запуска реестра API, как показано в листинге 9.7. (Обратите внимание: на этом шаге также запускается шлюз Kong, который будет использоваться позже в пайплайне.)

Листинг 9.7. Запуск реестра API в режиме фонового сервиса

```
    steps:
      - name: Build - Checkout
        uses: actions/checkout@v3
      - name: Build - Start background
➥ services (Kong gateway and Apicurio API registry)
        run: |
          docker compose -f chapter9/compose.yaml up -d
      - name: Build - Set up JDK 17
```

Добавляет новый шаг для запуска реестра API

Затем, после шага, на котором проводился линтинг API, добавьте в `build.yaml` шаги для архивации преобразованного файла определения API (чтобы затем загрузить его и просмотреть в случае необходимости), опубликуйте в реестре API и сохраните номер версии артефакта из реестра, как показано в листинге 9.8.

Листинг 9.8. Публикация определения API в реестре API

```
      - name: Build - Run API linting
        run: |
          npx @stoplight/spectral-cli lint build-output/api.yaml
➥ --ruleset chapter8/apis/rulesets/json-api.ruleset.yaml
      - name: Build - Archive artifacts
        uses: actions/upload-artifact@v3
        with:
          name: dist-after-linting
          path: |
            build-output
      - name: Build - Publish updated API
➥ definition to registry
        run: |
          curl -v -X POST
➥ 'http://localhost:9090/apis/registry/v2/groups/acmeapis
➥ /artifacts?ifExists=UPDATE'    \
           --header 'Content-Type: application/x-yaml'     \
           --header 'X-Registry-ArtifactId: productsapi'    \
           --header 'X-Registry-ArtifactType: OPENAPI'     \
           --data-binary "@build-output/api.yaml"    \
           --output created-artifact.json   \
           --fail-with-body
      - name: Build - Get created artifact version
        id: artifact-version
        run: |
          echo "ARTIFACT_VERSION=$(cat created-artifact.json
➥ | jq -j '.version')"  >> $GITHUB_OUTPUT
```

Добавляет новый шаг для архивации артефактов

Добавляет новый шаг для публикации определения API в реестре API

https://liveinternet.club

Теперь сохраните файл `.github/workflows/build.yaml`, зафиксируйте изменения и создайте пул-реквест для слияния с главной веткой. При слиянии пул-реквеста вы увидите, как рабочий поток выполняется на вкладке Actions репозитория GitHub.

> **ПРИМЕЧАНИЕ** В этом примере я описываю только отправку определений OpenAPI в реестр API. Вы можете добавить шаг для генерации коллекций Postman на основе определения OpenAPI и их отправки в рабочее пространство Postman с использованием Portman CLI (https://github.com/apideck-libraries/portman); я упоминал об этой возможности в главе 7. В документации Portman объясняется, как это сделать. Попрактикуйтесь в таком упражнении после чтения этого раздела.

Пайплайн преобразовал файл определения API и успешно опубликовал его в реестре API. Теперь обсудим, как использовать его для имитации API.

9.2. Стадия сборки: симуляция API с использованием моков

Пока вы еще пишете код сервиса API, приложения ниже по потоку, интегрированные с вашим API, могут уже начать работу над интеграцией с API. Как предоставить им API, с которым они могут работать, если вы еще не завершили реализацию? Или, переформулируя задачу, как вести разработку и тестирование относительно зависимости API, которая еще недоступна или не контролируется вами?

Проблема решается симуляцией (simulation) API. Модели API имитируют реальные API, принимая запросы и возвращая ответы, которые позволяют работать над приложениями, зависящими от таких API, и тестировать их. Схема процесса изображена на рис. 9.3. Без симуляции API потребителям API пришлось бы приостановить разработку, пока зависимость API не станет доступной.

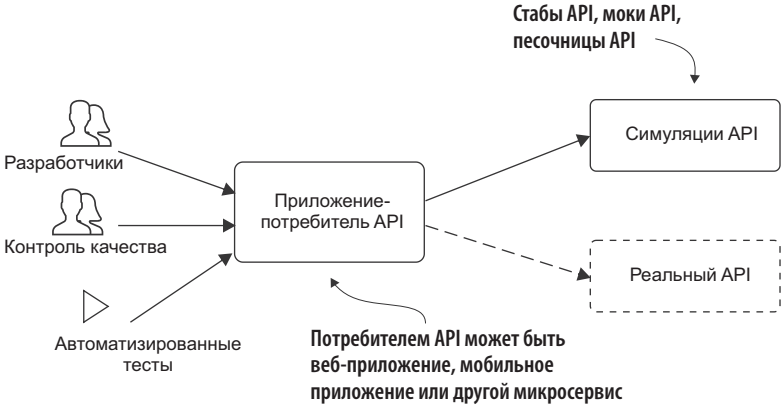


Рис. 9.3. Симуляции API заменяют реальные API для поддержки разработки и тестирования

https://liveinternet.club

Модели API можно классифицировать следующим образом.

- *Стабы (stub, заглушки)*, или *пустышки (dummy), API* — простые модели сервера API с ограниченной функциональностью. Они создаются для возвращения заранее определенных ответов на клиентские запросы или одинаковых ответов на все запросы. Это может быть полезно на раннем этапе тестирования интеграции API, когда основное внимание уделяется созданию рабочей интеграции для приложений на стороне клиента (вместо тестирования всех взаимодействий и сценариев сбоев).
- *Моки API* — более совершенная модель поведения реального API. Может настраиваться для возвращения динамических ответов на основании заданных условий, что позволяет разработчикам тестировать различные сценарии успешных и ошибочных взаимодействий при более точном моделировании условий реального мира.
- *Песочницы API* (sandbox) — фактическая реализация API, работающая в контролируемой изолированной среде, которая используется для исследования, тестирования и понимания того, как работает API без влияния на рабочие данные и системы. Провайдеры API обычно предоставляют песочницам API подмножество функциональности полного API и могут давать разработчикам доступ к песочнице как часть онбординга и улучшения опыта взаимодействия. Примеры песочниц API — Uber API Sandbox (https://mng.bz/9djj) и eBay API Sandbox (https://mng.bz/jXWa).

Стабы и моки API могут создаваться на основе файла определения OpenAPI (например, путем извлечения определений, хранящихся в реестре API). Также им могут передаваться другие статические файлы (например, JSON или YAML), которые указывают, каких запросов ожидать и какие ответы на них предоставить. Они также могут поддерживать конфигурацию на программном уровне и способность записывать и использовать ответы от реальных API. В идеале стабы и моки API должны вести себя по возможности близко к реальному API, но их не должно быть слишком сложно создавать и обслуживать. После краткого введения в имитацию API я опишу ее ключевые преимущества.

Проектирование API по принципу «сначала мок»

Помимо поддержки разработки и тестирования, моки API могут пригодиться при создании прототипа на этапе проектирования API. Этот подход, называемый «сначала прототип» или «сначала мок» (mock-first), основан на том, что проектировщики API начинают работу над дизайном API с создания мока вместо определения OpenAPI. Проектировщики и потребители API совместными усилиями создают мок с применением соответствующих инструментов. Далее он используется как инструмент для построения начальной версии предполагаемого дизайна. Затем проектировщик API может сгенерировать определение OpenAPI на основании прототипа, созданного с помощью инструмента создания моков.

Подробнее о подходе «сначала мок» см. в посте «Mock-First API Prototyping» Тома Эйкхерста (Tom Akehurst) (https://mng.bz/WEng).

https://liveinternet.club

9.2.1. Преимущества симуляции API

Симуляция API предоставляет ряд преимуществ для разработчиков и тестировщиков.

- *Поддержка параллельной разработки.* Фронтенд-команды могут работать над интеграцией с API и тестировать свои приложения, используя симуляцию API, пока бэкенд-команды работают параллельно над программированием API.
- *Обеспечение доступности.* Симуляции API могут устранить зависимость от доступности и стабильности реального сервиса API в процессе разработки и тестирования потребительских приложений.
- *Помощь в изоляции тестов.* Автоматизированные тесты можно выполнять с моками API, что отделяет процесс тестирования от самих API, которые могут быть ненадежными, дорогими или иметь ограничения.
- *Поддержка тестирования реальных сценариев.* Моки API помогают моделировать различные тестовые сценарии и условия (например, высокую сетевую задержку или случайные сбои). Это делает возможным тестирование приложения в таких сценариях.
- *Сокращение затрат.* Моки API помогают сократить затраты на запросы API в ходе разработки и интеграции с дорогими сторонними API. Симуляция также снижает потребность в дополнительных аппаратных ресурсах для запуска реальных API.
- *Повышение эффективности тестирования.* Использование мока или стаба API в некоторых тестах может ускорить тестирование и снизить сетевую задержку.

Как видите, симуляция API обеспечивает ряд преимуществ. Существуют различные инструменты для создания моков, но необходимо понимать, какие именно виды моков API можно получить с их помощью. Рассмотрим эту тему в следующем разделе.

9.2.2. Типы моков API и режимы работы инструментов создания моков

Моки и стабы API создаются с помощью специальных инструментов. Существует много подобных инструментов с открытым исходным кодом, например: Prism (https://github.com/stoplightio/prism), Microcks (https://github.com/microcks/microcks), WireMock (https://github.com/wiremock/wiremock), Hoverfly (https://github.com/SpectoLabs/hoverfly).

Моки API можно классифицировать на основании того, где они выполняются.

- *Локальные моки* могут запускаться на рабочей станции разработчика. Это полезно в сценариях, где требуется гибкость для запуска локальных тестов, низкая задержка или автономный доступ.

https://liveinternet.club

- *Серверные моки* (также называемые *общедоступными* (public), *размещенными* (hosted) или *облачными*) развертываются на общедоступных серверах (например, на внешних платформах SaaS (Software-as-a-Service)) или на внутренних серверах, к которым могут обращаться разные люди, команды или приложения.

Моки API также могут классифицироваться по имитируемым ими API.

- *Внутренние моки API*, которые принадлежат вам.
- *Внешние моки API* — моки сторонних API, которые вы не контролируете. Внешние моки API полезны, когда сторонние API, от которых вы зависите, предоставляют только фиксированную квоту запросов, которую вы с большой вероятностью превысите, имеют дорогостоящий план использования либо оказываются недоступными, нестабильными или трудными в предоставлении.
- В зависимости от типа симуляции API, которую вы намерены создать, инструменты создания моков API могут работать в разных режимах. Например, в *статическом* режиме, или режиме *ответа-заглушки*, когда для запросов в каждому эндпоинту возвращаются заранее заготовленные ответы HTTP. Недостаток этого режима — однообразие и устаревание данных. Чтобы опробовать этот режим, запустите локальный статический мок в Prism такой командой:

```
prism mock chapter8/apis/product-catalog.oas.yaml
```

Отправьте следующий запрос к эндпоинту в определении API и обратите внимание, что вы всегда получите один и тот же ответ:

```
curl --location 'http://localhost:4010/v1/catalog/categories'
➥ --header 'Accept: application/vnd.api+json'
➥ --header 'Authorization: Bearer my-secret-token'
```

Также инструменты создания моков могут работать в режиме *динамического ответа*, когда ответы генерируются динамически. Чтобы опробовать эту возможность, выполните команду, которая запускает Prism в динамическом режиме:

```
prism mock --dynamic chapter8/apis/product-catalog.oas.yaml
```

Повторите запрос `curl` и обратите внимание, что, хотя схема или форма ответа остается неизменной, сам ответ каждый раз разный.

Некоторые инструменты создания моков (например, WireMock и Hoverfly) также поддерживают другие режимы, такие как «запись — воспроизведение» и режим прокси. В режиме «запись — воспроизведение» инструмент создания моков работает в режиме прокси-сервера, перехватывая и прозрачно записывая запросы и ответы между клиентским приложением и API. При последующих запусках инструмент может работать в статическом режиме и возвращать сохраненные ответы на соответствующие запросы. В режиме прокси инструмент создания моков предоставляет ответы-заглушки для запросов, которые определены в актуальном API, и пропускает остальные запросы.

https://liveinternet.club

Ранее я упоминал о таком преимуществе моков API, как упрощение тестирования в реальных условиях. В следующем разделе мы рассмотрим пример.

9.2.3. Использование моков API для тестирования реальных сценариев

Представьте, что вам требуется протестировать поведение потребителя API при случайных задержках времени ответа. Допустим, вы хотите имитировать равномерное распределение задержки в интервале от 50 до 800 миллисекунд. Внедрить такое поведение и воспроизвести его по требованию в реальных условиях может быть сложно, но это можно сделать с помощью мока API. Ниже рассматривается пример такого рода в WireMock — открытом инструменте, который (среди прочего) поддерживает создание ответов мока в файле JSON.

В папке `chapter9/mapping` создайте для WireMock файл соответствий JSON с именем `test-response.json`. В файле перечислите запросы, для которых должен подбирать ответы WireMock, а также задайте задержку, которую необходимо имитировать (см. листинг 9.9). Эта конфигурация указывает WireMock возвращать ответ «Hello world» для запросов к `GET /some/endpoint`.

Листинг 9.9. JSON-файл соответствий WireMock

```
{
  "request": {
    "method": "GET",
    "url": "/some/endpoint"
  },

  "response": {
    "status": 200,
    "delayDistribution": {
      "type": "uniform",
      "lower": 50,
      "upper": 800
    },
    "body": "{ \"message\": \"Hello world!\" }",
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```

Задает равномерное распределение задержки с верхней и нижней границами

Теперь запустите WireMock следующей командой на порте 8080 в контейнере `docker` и выполните поиск файлов соответствий в каталоге `chapter9/mappings`:

```
cd chapter9
./start-wiremock.sh
```

Отправьте несколько запросов `GET` к http://localhost:8080/some/endpoint и обратите внимание на время ответа. Также можно использовать предоставленный скрипт

https://liveinternet.club

`curl chapter9/request.sh`, который возвращает общее время ответа в секундах. Он запускается так:

```
$>./chapter9/request.sh
{ "message": "Hello world!" }
[ Total time: 0.733438s ]
$>./chapter9/request.sh
{ "message": "Hello world!" }
[ Total time: 0.109310s ]
$>./chapter9/request.sh
{ "message": "Hello world!" }
[ Total time: 0.280190s ]
```

Как видите, время ответа API соответствует заданным границам. Потребители этого эндпоинта теперь могут проверить, как будут отвечать их приложения, когда в API возникают подобные задержки.

Итак, мы обсудили, как преобразовывать определения API в пайплайне развертывания, сохранять их в реестре API и создавать моки API для разработки и тестирования. А теперь рассмотрим еще одну задачу, которая может быть актуальной на этапе сборки, — генерацию SDK для API.

9.3. Этап сборки: генерация и хранение SDK

Представьте, что вашим потребителям API требуется Java SDK, потому что это позволит им быстро начать интеграцию с API. Как сгенерировать его в пайплайне?

Существуют разные способы генерации клиентского SDK на этапе сборки пайплайна. Можно воспользоваться генератором SDK (например, OpenAPI Generator CLI, рассмотренным в главе 6) для генерации DK по собранному файлу определения OpenAPI. Также можно воспользоваться провайдерами CGaaS (CodeGen-as-a-Service), которые подключаются к пайплайну развертывания для генерации многоязыковых SDK. Среди примеров можно отметить Speakeasy (www.speakeasyapi.dev) и APIMatic (www.apimatic.io).

При генерации SDK артефакт необходимо сохранить в месте, где он может использоваться другими командами и приложениями, когда он понадобится. В пайплайне развертывания генерируемые артефакты обычно сохраняются в *репозиториях двоичных артефактов* — хранилищах, способных сохранять текст и двоичные артефакты (например, программные пакеты) вместе с их метаданными.

Желательно, чтобы провайдеры API предоставляли потребителям многоязыковые SDK. SDK улучшают опыт взаимодействия разработчика, а также предоставляют документацию API и ускоряют интеграцию для потребителей API. Это способствует более широкому принятию и интенсивному использованию продукта API. Вспомогательные классы и помощники по аутентификации,

https://liveinternet.club

предоставляемые SDK, ускоряют потребление API. Языковые привязки позволяют разработчикам напрямую использовать объекты и функции языка для взаимодействия с API. Я перечислю некоторые преимущества предоставления SDK провайдерами API.

9.3.1. Преимущества предоставления SDK

SDK улучшают опыт разработчика по двум причинам.

- Без SDK бремя создания функций для вызова API на своем языке программирования, применения правильных стандартов кодинга и создания качественных SDK ложится на потребителей API. Им приходится поддерживать и обновлять эти SDK при обновлении API.
- SDK предлагают уровень абстракции, упрощающий их использование. Некоторые операции требуют нескольких вызовов API, и надлежащим образом построенные SDK могут абстрагировать эту необходимость, чтобы потребителю API было достаточно вызвать всего одну функцию. Хороший пример — необходимость обновления токена аутентификации при отправке запроса. Без заранее построенного SDK потребители API должны определить, что срок действия токена истекает, и инициировать вызов обновления токена, прежде чем переходить к нужному вызову API. SDK может абстрагировать эту необходимость и скрыть эту сложность от потребителей. В некоторых случаях абстракция, предоставляемая SDK, может способствовать снижению количества запросов, так как потребители API получают более простой и удобный интерфейс SDK, с которым проще работать, чем с низкоуровневыми вызовами REST.

Организациям, создающим продукты «API-first» (то есть платформы, где API является главным интерфейсом интеграции), стоит рассмотреть возможность предоставления клиентских SDK. Организациям, которые не создают такие продукты, стоит оценить, насколько обоснованными будут вложения в создание SDK.

Принимая решение о том, стоит ли предлагать SDK на конкретном языке программирования, подумайте, способствует ли это более частому выбору вашего продукта потенциальными покупателями. Кроме того, оцените, готовы ли вы нести ответственность за создание и поддержку того улучшенного опыта разработчика, который обеспечивают SDK, или же разработчики должны будут полагаться на сторонние решения либо альтернативы с открытым исходным кодом.

Вы увидели, как SDK упрощают работу разработчика и повышают ее удобство. Теперь обсудим проблемы, с которыми вы можете столкнуться при создании SDK.

9.3.2. Проблемы при создании SDK

Создание и обслуживание SDK влечет для провайдеров API следующие сложности.

https://liveinternet.club

- *Дополнительные затраты на обслуживание*. Создание SDK для API на конкретном языке программирования требует определенных начальных вложений. Они включают затраты на обновление пайплайна разработки для поддержки генерации, тестирования и документирования новых SDK. Существуют и накладные расходы на решение конкретных проблем, связанных с этим языком программирования и исправлением ошибок.
- *Выделение ресурсов*. В каждом языке программирования есть свои стандарты написания кода и идиоматика. Поставщикам API неоходимо обладать экспертизой в каждом языке, для которого они предоставляют SDK, чтобы гарантировать следование лучшим практикам. Таким образом, поддержка нескольких SDK на разных языках требует времени и ресурсов персонала, чего в небольших организациях может и не быть.
- *Управление релизом*. Новая версия API может нарушить совместимость со старыми версиями SDK; следовательно, вам понадобится новая версия SDK, которую необходимо тщательно протестировать перед релизом. Убедить потребителей API, пользующихся старыми версиями SDK, перейти на новую может быть непросто. Отслеживание версий SDK и языков, на которых работают разные клиенты, может быть сложной задачей в масштабе.

У предоставления клиентам SDK, поддерживаемых организацией, существуют альтернативы. Одна из них — делегирование создания SDK потребителю API, для чего ему просто передается файл определения и документация API. Другой возможный вариант — создание SDK с открытым исходным кодом и его релиз с пометкой о том, что SDK публикуется открытым сообществом. Сообщество также может предоставить альтернативные SDK на основании определения API. У этих решений есть свои недостатки: они могут привести к тому, что организация утратит контроль над формированием и поддержкой интеграции со стороны разработчиков.

Мы рассмотрели преимущества и недостатки создания SDK. Теперь обсудим, как генерировать их в пайплайне из нашего примера.

9.3.3. Обновление пайплайна для генерации SDK

Я покажу, как подключить OpenAPI Generator CLI к платформе GitHub Actions и сохранить результат в GitHub Packages — репозитории двоичных артефактов GitHub для хостинга и управления программными пакетами и контейнерами. Таблица 9.2 кратко описывает, почему следует генерировать SDK, а также содержит другие важные моменты.

Прежде чем начать, обращу ваше внимание, что каталог `chapter8/java-template` содержит шаблон генератора OpenAPI для генерации клиентского SDK для Java на базе клиента `okhttp-gson`.

https://liveinternet.club

Таблица 9.2. Сводка генерируемых Java SDK

Тип артефакта	Пользователи артефакта	Для чего нужен артефакт	Исходные файлы, на основе которых генерируется артефакт	Инструмент для генерации артефакта	Где будет храниться артефакт
Клиентские SDK для Java	• Внешние потребители API • Внутренние приложения, потребляющие API	Внутренние приложения, интегрируемые с внешним API	Файлы определений API	OpenAPI Generator	GitHub Packages

Альтернативный формат определения API для генерации SDK: Fern

В нашем примере SDK генерируется на основе определения OpenAPI. Однако это не единственный вариант. Fern (https://buildwithfern.com) — инструмент с открытым исходным кодом для создания REST API, и формат определения Fern API является альтернативой для формата определений OpenAPI. В поставку Fern включен компилятор, который может генерировать клиентские SDK, заглушки кода на стороне сервера, коллекции Postman и даже файл определения OpenAPI. Определения Fern, как и определения OpenAPI, записываются в формате YAML. Однако Fern стремится к упрощению записи, что позволяет избежать неоднозначностей, с которыми сталкиваются генераторы кода из-за неразличаемых объединений OpenAPI и ключевого слова `anyOf`. Кроме того, компилятор ориентирован на генерацию идиоматического кода. Чтобы больше узнать о сравнении Fern с OpenAPI, посетите сайт Fern (https://mng.bz/8w9K).

СОВЕТ Возможно, вас интересует, как был создан клиентский шаблон `chapter8/java-template`. Как упоминалось в главе 6, чтобы сгенерировать шаблон OpenAPI, который можно будет изменить, следует воспользоваться командой `openapi-generator-cli author template`. Другой способ — скопировать шаблон из исходной папки OpenAPI Generator. Для нашего Java SDK это означает копирование папки https://mng.bz/EZrq. Я сделал это за вас и добавил необходимый файл шаблона в каталог `chapter8/java-template`, но вы можете найти дополнительную информацию на странице https://openapi-generator.tech/docs/templating.

Теперь добавьте шаг генерации SDK, в котором OpenAPI Generator CLI используется для генерации SDK на основе преобразованного файла определения API (листинг 9.10).

Сохраните это изменение в своей ветке, создайте пул-реквест, слейте с основной веткой и посмотрите, как выполняется сборка. На этом шаге вы сгенерировали SDK. Но где должен храниться сгенерированный артефакт? Ответим на этот вопрос в следующем разделе.

https://liveinternet.club

Листинг 9.10. Добавление шага для генерации Java SDK

```
- name: Build - Generate Java SDK
        run: |
          sed -i "s/OWNER/${{ github.repository_owner }}/g"
➥ chapter8/java-template/libraries/okhttp-gson/pom.mustache
          sed -i "s/REPO_NAME/${{ github.event.repository.name }}/g"
➥ chapter8/java-template/libraries/okhttp-gson/pom.mustache
          tail -8  chapter8/java-template/libraries/
➥ okhttp-gson/pom.mustache
          npm install -g @openapitools/openapi-generator-cli
          openapi-generator-cli generate
    ➥ --input-spec build-output/api.yaml
    ➥ --generator-name java
    ➥ --output build-output/java-client
    ➥ --template-dir chapter8/java-template
    ➥ --library okhttp-gson
    ➥ --additional-properties=invokerPackage
    ➥ =com.acmepetsupplies.client,
    ➥ apiPackage=com.acmepetsupplies.api,
    ➥ modelPackage=com.acmepetsupplies.model,
    ➥ groupId=com.acmepetsupplies,
    ➥ artifactId=acmepetsupplies-java-client,
    ➥ artifactVersion=1.0.0-SNAPSHOT
```

Путь к файлу спецификации OpenAPI

Использует генератор клиентского кода Java

Выводит каталог для сгенерированного кода

Путь к измененному каталогу шаблона

Использует клиент OkHttp с сериализацией GSON

Дополнительные свойства для настройки генерируемого кода

Определение репозитория Maven для GitHub Packages

Обратите внимание: в только что написанном нами коде первая пара строк начинается с использованием команд Sed для замены `OWNER` именем учетной записи GitHub, а `REPOSITORY` — именем репозитория GitHub в файле `pom.mustache`. Файл `pom.mustache` — шаблон, используемый для создания файла `pom.xml` инструментом OpenAPI Generator. Значения `OWNER` и `REPOSITORY` встречаются в URL, заданном в виде https://maven.pkg.github.com/OWNER/REPOSITORY в элементе `<distributionManagement>`. Этот элемент сообщает Maven, где должен публиковаться пакет — задача, которой мы займемся в следующем разделе.

9.3.4. Хранение SDK в репозиториях артефактов

На CI-этапе пайплайна развертывания генерируются артефакты, к числу которых относятся откомпилированный код, развертываемый пакет релиза-кандидата, SDK и т. д. Их можно хранить в системе контроля версий, но это неэффективно, так как системы контроля версий предназначены для управления текстовыми, а не двоичными файлами. Репозиторий двоичных артефактов служит централизованным местом для хранения артефактов, генерируемых на стадии сборки CI-этапа пайплайна, чтобы далее, на CD-этапе, не приходилось генерировать артефакты заново. Их можно будет просто взять из репозитория. Схема процесса показана на рис. 9.4. Примеры менеджеров репозиториев артефактов — Sonatype Nexus, JFrog Artifactory и GitHub Packages.

https://liveinternet.club

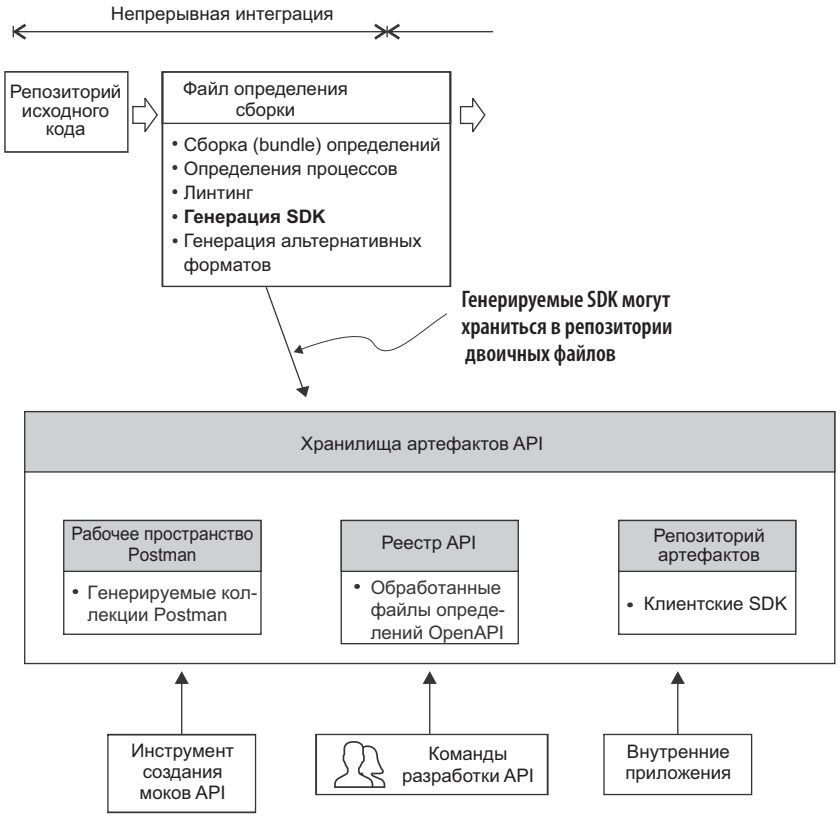


Рис. 9.4. Обновление пайплайна для генерации SDK

Репозитории артефактов обеспечивают следующие преимущества.

- *Совместная работа и управление зависимостями*. Репозитории артефактов предоставляют центральную точку для совместного использования артефактов с другими командами. В организациях могут существовать разные серверы CI, генерирующие артефакты. Репозиторий может служить центром для публикации этих артефактов, чтобы команды и более поздние этапы пайплайна развертывания могли обращаться к ним. Репозитории артефактов могут хранить программные пакеты и зависимости, необходимые для процесса сборки.
- *Эффективная сборка*. После создания артефакта пайплайн не придется запускать заново для этой же цели, так как последующие этапы пайплайна, которым понадобится артефакт, смогут загрузить его из репозитория.
- *Версионирование артефактов и метаданных*. Репозиторий может сохранять каждый артефакт как новую версию. Он также может хранить метадан-

https://liveinternet.club

ные (например, хеш артефакта, время создания и другую информацию). Кроме того, при публикации в репозитории артефакт и его метаданные не изменяются. Это может быть полезно для выявления и отладки проблем в прошлых релизах продукта. Достаточно загрузить пакет артефакта, для которого было получено сообщение об ошибке, и отладить его.

- *Постепенное продвижение.* Артефакты можно продвигать (например, перемещая их в другую папку в репозитории артефактов) для перехода между разными средами организации (средами разработки, контроля качества, промежуточной средой и продакшен). В этом случае не придется хранить в репозитории все сгенерированные артефакты. Можно ограничиться только теми, которые успешно прошли все этапы пайплайна, и уничтожить остальные.
- *Контроль доступа.* Репозитории артефактов также предоставляют средства контроля доступа, чтобы помочь управлять разрешениями на публикацию артефактов или их извлечение из репозитория.

Итак, вы узнали о преимуществах хранения SDK в репозитории артефактов. Теперь я покажу, как поместить сгенерированные SDK в репозиторий.

9.3.5. Обновление пайплайна для сохранения SDK

Добавьте в файл `.github/workflows/build.yaml` шаг для публикации сгенерированных SDK в GitHub Packages, как показано в листинге 9.11. Команда Maven `deploy` публикует файл `.jar` для SDK в удаленном репозитории пакетов вместе с другими метаданными, файлами хешей SHA1 и MD5.

Листинг 9.11. Шаг публикации Java SDK

```
- name: Build - Publish SDK to GitHub Packages
  run: |
    cd build-output/java-client
    mvn --batch-mode deploy
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Развертывает сгенерированный код в GitHub Packages

Токен GitHub для аутентификации в GitHub Packages

Теперь создайте пул-реквест со своими изменениями, проведите их коммит и слияние. Вы увидите, как выполняется сборка, а сгенерированный SDK публикуется в репозитории артефактов. В правой части главной страницы репозитория GitHub должна появиться секция Packages со ссылкой на сгенерированный пакет: `com.acmepetsupplies.acmepetsupplies-java-client 1.0`.

> **СОВЕТ** Чтобы увидеть, как должен выглядеть завершенный рабочий поток, см. файл `.github/work-flows/build-transform.yaml.completed`.

Когда артефакты API будут сгенерированы, можно переходить к этапу CD пайплайна — развертыванию API в шлюзе в промежуточных средах.

https://liveinternet.club

9.4. Этап развертывания и тестирования: развертывание конфигурации API-шлюза

Представьте, что вы построили фоновый сервис API и теперь хотите развернуть его и предоставить внешний доступ к нему через API-шлюз в одной из сред.

Допустим, вы хотите дать доступ только к одному эндпоинту вашего API — `GET /v1/catalog/products/<uuid>`. На рис. 9.5 показано, на каком этапе пайплайна должна развертываться конфигурация.

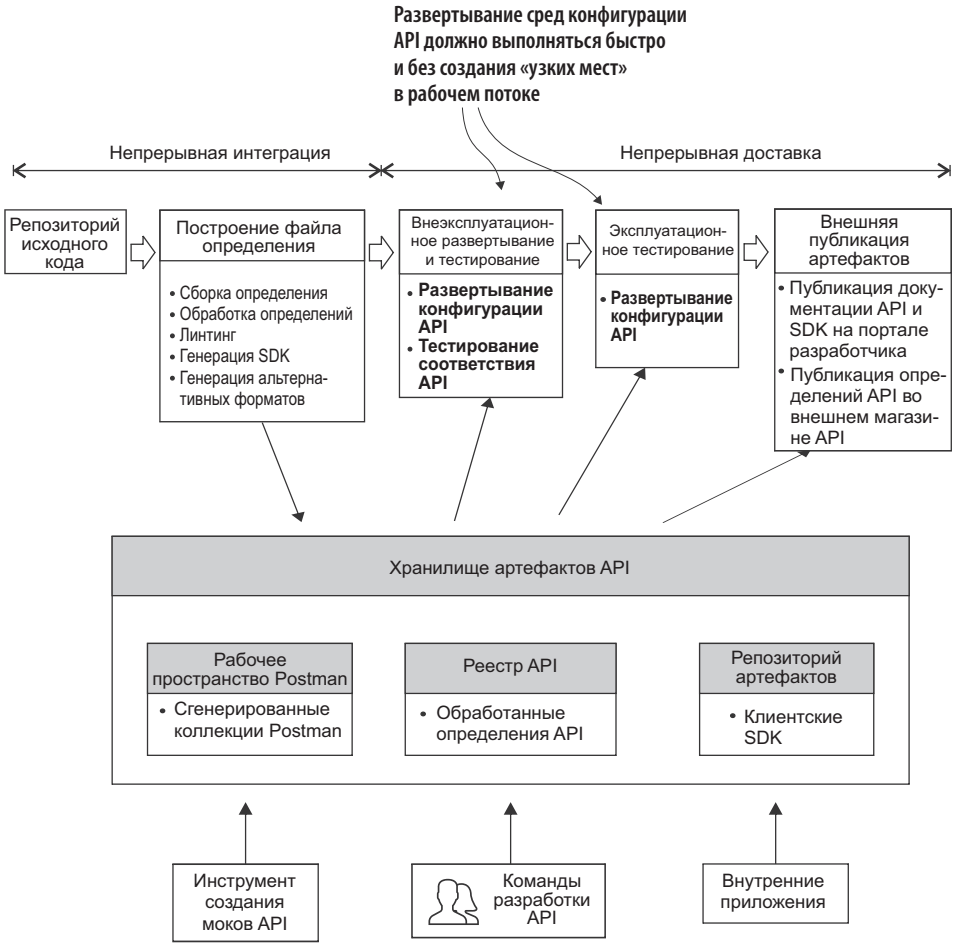


Рис. 9.5. Развертывание конфигурации API в разных средах в пайплайне

Допустим, ваш API — всего лишь один из многих, которые должны предоставляться центральным API-шлюзом вашей организации (шлюзом управляет

https://liveinternet.club

команда платформы API). Как создать конфигурацию, дающую доступ к API без необходимости создавать рабочий тикет для команды, которая управляет API-шлюзом? Иначе говоря, как избежать создания «узкого места» в рабочем потоке, которое возникает из-за передачи работы другой команде, и снизить время исполнения доставки?

Ответ на этот вопрос занимает центральное место в методологии APIOps, предоставляющей механизм автоматизации развертывания конфигурации API в API-шлюзе с использованием принципов GitOps. В следующем разделе я покажу, как это работает.

9.4.1. Развертывание APIOps

В рамках APIOps все изменения в состоянии конфигурации осуществляются обновлением файла состояния конфигурации в системе контроля версий (VCS). Это рабочий поток GitOps, применяемый к конфигурации API. Вспомним базовые принципы модели GitOps, о которых шла речь в главе 1.

- *Декларативная конфигурация* — желаемое состояние управляемой системы (то есть все данные конфигурации, необходимые для воссоздания экземпляра системы) указывается декларативно. Декларативные конфигурации описывают ожидаемое операционное состояние, не указывая, как оно будет достигнуто. Этот подход отделяет ответственность определения конфигурации от ее реализации. Под реализацией подразумеваются императивные команды, вызовы API и скрипты, которые выполняются для достижения желаемого состояния.
- *Версионированная и неизменяемая конфигурация* — желаемое состояние сохраняется в виде неизменяемых версий. Используемое хранилище содержит полную историю версий объявлений состояния, а также предоставляет контроль доступа и аудит изменений в желаемом состоянии. Git — канонический пример хранилища состояния. Такие механизмы, как пул-реквесты и владельцы кода (CODEOWNERS), могут использоваться для утверждения изменений в желаемой конфигурации состояния.
- *Автоматическое извлечение конфигурации* — автоматизированные процессы, называемые *программными агентами*, извлекают объявление желаемого состояния из источника, в котором оно хранится (обычно из системы контроля версий Git). Программный агент может быть как автономным процессом, так и пайплайном развертывания.
- *Непрерывное согласование конфигурации* — программные агенты непрерывно согласовывают фактическое состояние системы с желаемым. Иначе говоря, они наблюдают за фактическим состоянием системы для выявления любых отклонений от желаемого (также называемых *дрейфом*). При обнаружении таких отклонений агенты пытаются обновить текущее состояние, чтобы оно совпадало с желаемым.

https://liveinternet.club

GitOps не приветствует ручные изменения (например, через пользовательский интерфейс API-шлюза), поскольку они сопряжены с повышенным риском ошибок, создают сложности в отладке и у них может не быть той истории аудита, которую предоставляет журнал контроля версий. По сути, GitOps объединяет подходы «инфраструктура как код» (IaC, Infrastructure as Code), пул-реквестов и CI/CD-пайплайнов. IaC — это практика поддержания декларативной конфигурации инфраструктуры в системах контроля версий (VCS). GitOps следует этому шаблону: хранение конфигурации состояния в VCS и использование пул-реквестов для управления изменениями в ней. Пайплайн развертывания подхватывает эти изменения и развертывает их в системах, работающих в различных средах.

Существует ряд особенностей, которые должны учитывать команды, работающие по указанной модели.

- Команде разработки не нужно заводить рабочий тикет, чтобы команда платформы API создала конфигурацию API. Другими словами, никакая передача работы по конфигурации не требуется. Команда разработки действует в режиме самообслуживания: она самостоятельно создает конфигурацию и оформляет пул-реквест, запускающий проверку этой конфигурации, а ревью может выполнить команда платформы. После слияния с главной веткой процесс развертывания начинается автоматически. Такая модель ускоряет развертывания. Каждая команда разработки функциональности может передавать обновления несколько раз в день, развертывать конфигурацию API и быстро видеть результаты.
- Команда разработки не нуждается в пользовательском интерфейсе API-шлюза для обновления конфигурации API. Все делается через VCS. Кроме того, командам разработки не требуется прямой доступ к среде, в которой работает API-шлюз, для внесения изменений в его конфигурацию — им нужен доступ только к репозиторию конфигурации среды. Таким образом повышается безопасность развертывания.
- Команда имеет доступ к истории версий всех изменений конфигурации в VCS. Полная история изменений среды хранится в репозитории контроля версий. Файл состояния конфигурации содержит полное описание состояния API-шлюза, а сообщения в журнале коммитов объясняют, как оно изменялось со временем. Таким образом команда получает четкую документацию по состоянию развертывания. Отмена изменения может сводиться к выполнению команды `git revert` и наблюдению за процессом применения изменений. Запись в VCS становится как историей аудита, так и транзакционным журналом изменений в этой среде, обеспечивающим возможность отмены и возврата к любой точке.

СОВЕТ Чтобы узнать больше о преимуществах GitOps, я рекомендую статью «GitOps: A Path to More Self-Service IT» Томаса А. Лимончелли (Thomas A. Limoncelli) (https://queue.acm.org/detail.cfm?id=3237207).

https://liveinternet.club

Существуют два способа выполнения развертывания в GitOps: push-развертывание (развертывание проталкиванием) и pull-развертывание (развертывание извлечением). Оба способа объясняются в следующих подразделах.

Push-развертывание

В рамках этой модели каждый раз, когда разработчик выполняет коммит кода в репозитории приложения, запускается пайплайн развертывания, который строит контейнер приложения. Он также обновляет репозиторий конфигурации, добавляя туда манифесты развертывания для приложения и другие конфигурационные файлы (например, для API-шлюза). Обычно шаблоны манифестов развертывания хранятся в репозитории приложения, а пайплайн развертывания использует шаблоны для генерации реальных файлов манифестов, которые затем сохраняются в репозитории конфигурации. Когда в главной ветке репозитория конфигурации выполняется слияние изменений, пайплайн развертывания активируется и применяет обновленные манифесты к работающей системе в заданной среде (рис. 9.6). В примерах, которые были рассмотрены ранее в нашей книге, используется push-развертывание.

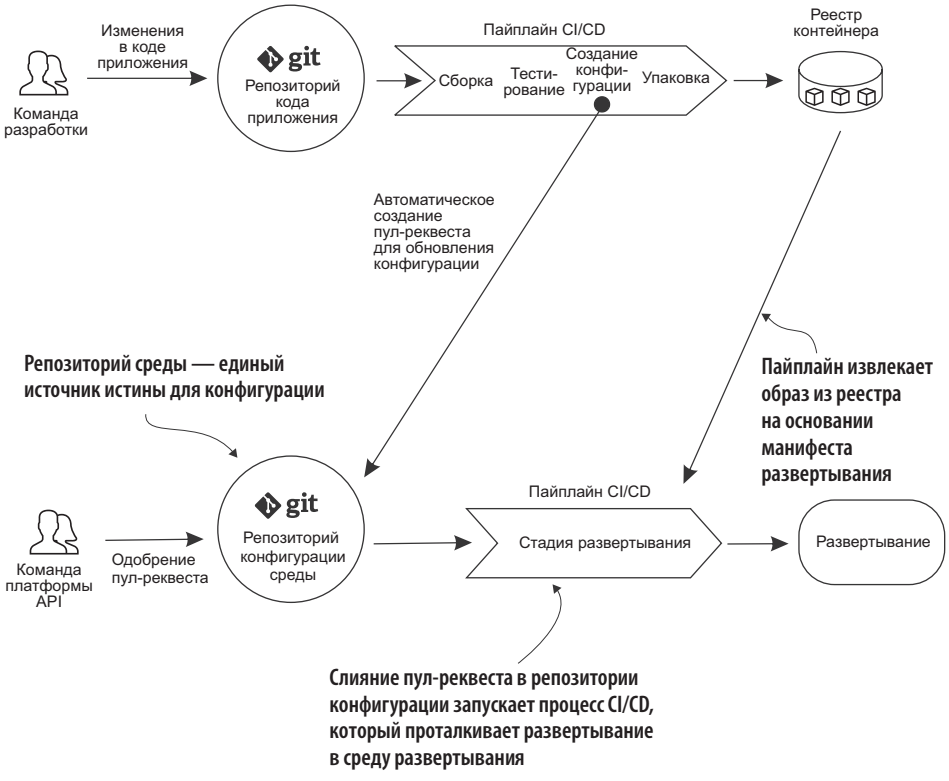


Рис. 9.6. Push-развертывание в GitOps

https://liveinternet.club

У этого подхода есть два ограничения. Во-первых, он требует, чтобы учетные данные целевой среды хранились в системе CI/CD, чтобы она могла получить доступ к целевой среде и обновить файлы конфигурации. Хотя в некоторых случаях это приемлемо, подобное обстоятельство может вызвать проблемы безопасности, поскольку в случае взлома системы CI/CD злоумышленник способен получить доступ к учетным данным целевых сред.

Во-вторых, развертывание целевого состояния активируется только изменениями в репозитории среды. Если кто-то войдет в пользовательский интерфейс API-шлюза и изменит конфигурацию вручную, может возникнуть расхождение между целевым состоянием, хранящимся в репозитории конфигурации, и фактическим состоянием в API-шлюзе. Расхождение может быть замечено не сразу, если только в системе не был настроен процесс мониторинга для постоянной проверки отсутствия дрейфа. Например, инструмент командной строки decK, использованный в первом примере, поддерживает команду `diff`, которая определяет отличия текущего состояния конфигурации в рабочем экземпляре Kong от заданного файла состояния конфигурации. В шлюзах, работающих в кластере Kubernetes, для просмотра различий в конфигурациях также может использоваться команда `kubectl diff`.

Pull-развертывания

При таком подходе *оператор* (operator) непрерывно отслеживает репозиторий Git на предмет изменений. Оператор GitOps — программный инструмент, который постоянно наблюдает за системой и поддерживает ее синхронизацию с источником желаемой конфигурации (обычно репозиторием Git). Когда оператор обнаруживает различие, вызванное либо изменением желаемой конфигурации в репозитории, либо изменением в работающей системе (например, из-за ручного изменения конфигурации через пользовательский интерфейс системы), он извлекает (pull) обновленное состояние конфигурации из репозитория Git и применяет его к целевой среде. Оператор работает в той же среде или кластере, что и развернутый шлюз. Поэтому ему не нужны учетные данные для обращения к среде, что повышает безопасность по сравнению с моделью push-развертывания. Примеры таких операторов для Kubernetes — Argo CD и Flux. На рис. 9.7 изображена схема pull-развертывания в GitOps.

Мы рассмотрели, как работает развертывание в средах. В следующем разделе обсудим, как API распространяются по средам.

9.4.2. Обновление конфигурации приложения

Обычно в корпоративных средах работает много сервисов приложений, каждый из которых имеет собственный репозиторий. Эти сервисы и связанные с ними конфигурации API должны распространяться по средам — например, среде разработки, контроля качества, промежуточной среде и продакшен. При обновлении конфигурации приложений необходимо учитывать два обстоятельства.

https://liveinternet.club

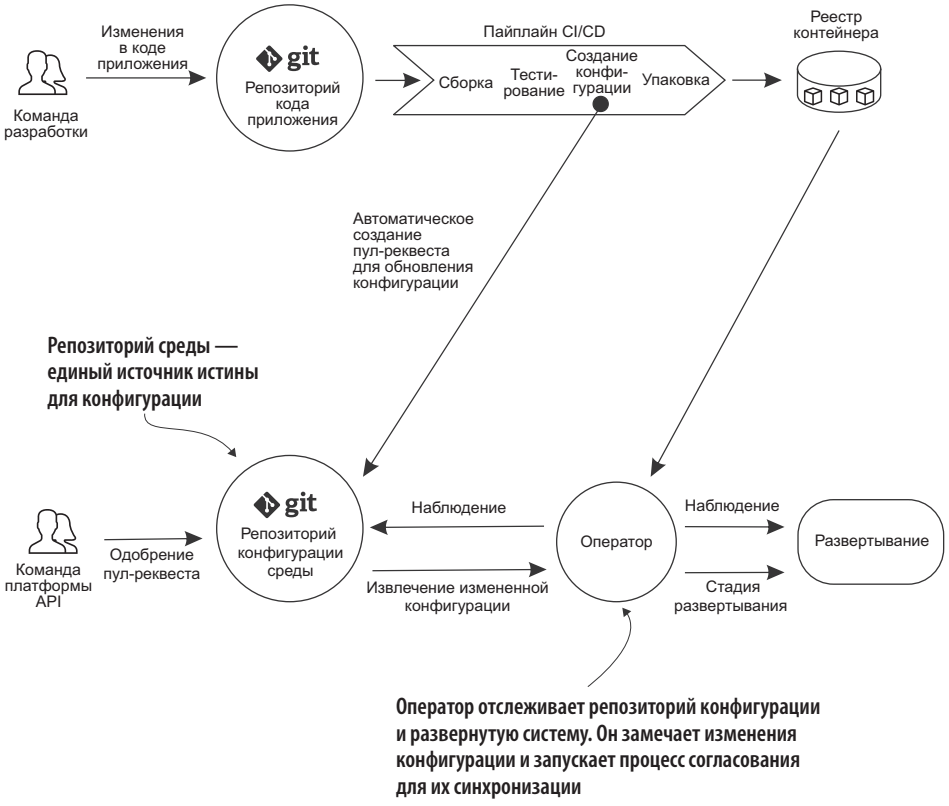


Рис. 9.7. Pull-развертывание в GitOps

Первое — как изменения в репозитории приложения могут инициировать обновление репозитория конфигурации (например, обновление конфигурации приложения, а в случае Kubernetes — обновление файла манифеста Kubernetes новой версией контейнера). Для этого с помощью шаблонов конфигурации приложения в репозитории приложения могут генерироваться фактическая конфигурация и файлы манифестов, которые проталкиваются в общие репозитории сред с использованием автоматизированного пул-реквеста. Репозиторий конфигурации содержит конфигурацию приложения и инфраструктуры для каждой среды. Когда происходит слияние пул-реквеста с главной веткой, пайплайн развертывания может развернуть изменения в необходимых средах.

Второе — как распространять конфигурацию между средами. Один из методов — сохранять разные конфигурации среды в разных ветках Git, но это связано со значительной сложностью при отслеживании изменений между ветками. Более простой способ основан на поддержании папки конфигураций для каждой среды. Распространение конфигураций включает копирование конфигурации

https://liveinternet.club

между двумя папками среды (рис. 9.8). Обратите внимание: некоторые команды могут решить поддерживать разные репозитории конфигурации вместо одного центрального, сводя к минимуму риск конфликтов пул-реквестов из разных пайплайнов, создающих автоматизированные пул-реквесты.

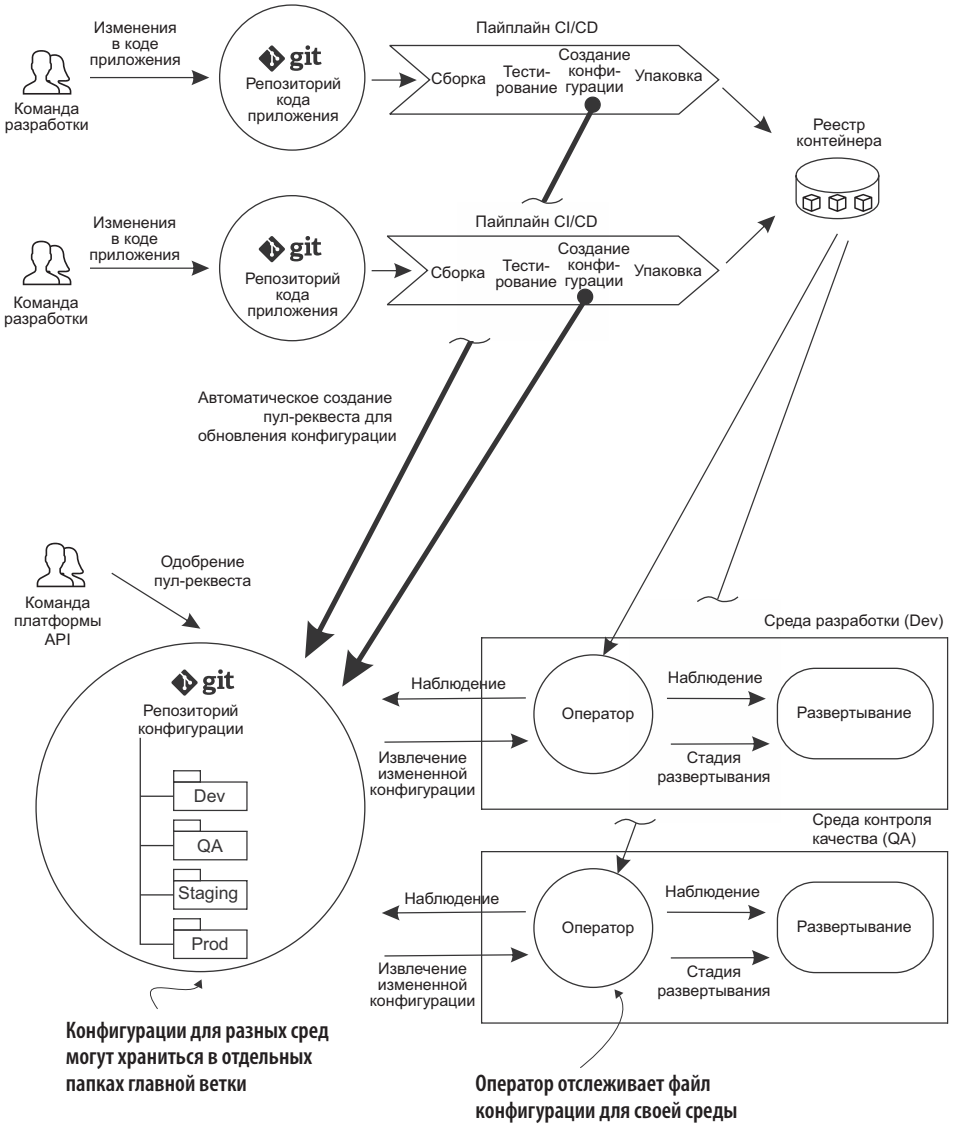


Рис. 9.8. Управление несколькими приложениями и средами с использованием pull-модели GitOps, с хранением конфигурации для разных сред в одном централизованном репозитории конфигурации

https://liveinternet.club

СОВЕТ Чтобы узнать больше об этом подходе, см. статью Костиса Капелониса (Kostis Kapelonis) «How to Model Your GitOps Environments and Promote Releases between Them» (https://mng.bz/NRrE).

Мы обсудили принципы GitOps, модели развертывания и подходы к обновлению файлов конфигурации. Но, как обычно, наряду с достоинствами у модели GitOps есть и недостатки.

9.4.3. Сложности GitOps

Принятие методологии GitOps сопровождается рядом сложностей.

- *Автоматизированные пул-реквесты с репозиториями конфигурации могут порождать конфликты*. На CI-этапе пайплайна создаются автоматизированные пул-реквесты для репозитория конфигурации каждый раз, когда выходит новая версия приложения или изменяется конфигурация приложения. В процессе работы над кодом это может происходить достаточно часто. Когда несколько таких CI-процессов приложения обновляют репозиторий конфигурации, может возникнуть конфликт из-за отсутствия синхронизации разных процессов, клонирующих один репозиторий (так как их локальная копия устаревает и им приходится повторять попытку извлечения/проталкивания). Проблема может быть решена внедрением сложного механизма повторных попыток создания пул-реквестов или разбиением репозитория конфигурации (например, по одному для каждого пространства имен) для минимизации конфликтов.
- *Бремя управления несколькими репозиториями Git*. Эта проблема связана с первой. С увеличением количества репозиториев конфигураций и приложений бремя управления репозиториями растет, так как репозитории необходимо настраивать с правильными правами доступа и соединять с агентами синхронизации (программными процессами или агентами сборки, синхронизирующими состояние кластера с желаемым состоянием конфигурации). Создание таких репозиториев отнимает время у разработки. Проблему можно решить созданием отдельного репозитория конфигурации на каждый кластер API-шлюза.
- *Отсутствие поддержки для централизованного управления секретами*. В корпоративных средах существует необходимость управления секретами (приватными ключами, паролями, сертификатами и т. д.) за пределами пайплайна развертывания — обычно в централизованном хранилище секретов, где можно провести их аудит. В GitOps нет специальных рекомендаций по управлению секретами. Хранить секреты в VCS исходного кода (таких, как Git) не рекомендуется, так как журнал Git хранит долговременную историю, и эта информация доступна всем разработчикам, клонирующим репозиторий. Любой, кто имеет доступ к репозиторию, сможет получить и доступ к секрету.

https://liveinternet.club

Обсуждение возможных решений этих проблем выходит за рамки книги, но введение в некоторые инструментальные решения можно найти в статье Адама Сандора (Ádám Sándor) «GitOps: The Bad and the Ugly» (https://mng.bz/Ddry).

Кроме проблем, свойственных GitOps, полезно рассмотреть, в какой степени GitOps как практика подойдет разным типам команд в стандартной организации. Поговорим об этом в следующем разделе.

9.4.4. APIOps и топологии команд

В своей работе «Team Topologies» (IT Revolution, 2019) Мэтью Скелтон (Matthew Skelton) и Мануэль Паис (Manuel Pais) описывают четыре фундаментальные разновидности команд, связанных с разработкой ПО: потоковые, платформенные, команды сложных подсистем и вспомогательные. Ниже приведены краткие описания каждого типа.

- *Потоковые команды* сосредоточены на одном потоке работы; они занимаются созданием продукта или сервиса, обладающего ценностью для клиентов. Чтобы быстро поставить результат, они должны работать по возможности независимо — то есть передавать другим командам минимально возможный объем работы. Потоковые команды — типичные функциональные команды, которые несут сквозную ответственность в потоке практической ценности, от сбора пользовательских требований и обратной связи от клиентов до создания программного продукта, его тестирования, поддержки на продакшен и исправления ошибок.
- *Платформенные команды* строят и поддерживают внутренние платформы (самообслуживаемые API, внутренние инструменты, документация, поддержка), которые используются потоковыми командами для доставки их продукта или сервиса в более высоком темпе и с меньшими усилиями по координации. Внутренняя платформа, над которой они работают, не является рыночным предложением организации; скорее она помогает минимизировать когнитивную нагрузку на потоковые команды, оптимизируя их рабочие потоки. Например, к платформенным командам относятся команды, поддерживающие API-шлюз, который используется потоковыми командами для предоставления доступа к API, и команды поддержки систем управления идентификационными данными, используемых для аутентификации и авторизации продукта API.

  СОВЕТ За обсуждением определения платформ обращайтесь к превосходной статье Эвана Ботчера (Evan Botcher) «What I Talk about When I Talk about Platforms» (https://martinfowler.com/articles/talk-about-platforms.html).

- *Вспомогательные команды* — это команды специалистов в технической области или области продукта, поддерживающие работу потоковых команд. Для этого они выполняют исследования и эксперименты с новыми техноло-

https://liveinternet.club

гиями и практиками, которые могут улучшить рабочий процесс потоковых команд. Они помогают потоковым командам, находящимся под давлением, устраняют препятствия на их пути и обнаруживают упущенные возможности. Они не владеют никакими программными компонентами, но действуют как группа экспертов-наставников, упрощающих работу потоковых команд. Пример вспомогательной команды — команда управления API, которая помогает развивать стандарты API организации, изучает автоматизированные инструменты управления API и поддерживает команды документацией и руководством о том, как быстро приступить к разработке API.

- *Команды сложных подсистем* занимаются созданием и обслуживанием сложных частей системы, требующих широких специальных знаний — например, алгоритмов или искусственного интеллекта. Это специалисты, работающие над подсистемами, требующими глубокого знания математики, теории вычислений или других технических знаний. Каждый член такой команды обычно является специалистом по конкретной подсистеме. Эти команды также помогают сократить когнитивную нагрузку на потоковые команды.

Применительно к процессам APIOps потоковые команды (ранее в книге я называл их *командами разработки*) отвечают за обновление конфигурации API (через пул-реквесты), необходимое для предоставления доступа к API. Платформенная команда — в данном случае команда платформы API — проводит ревью и одобряет пул-реквест для изменения конфигурации API. Платформенная команда также отвечает за подготовку или рекомендации по подготовке пайплайнов рабочего потока APIOps, инструментов синхронизации конфигурации и операторов. Они также следят, чтобы пул-реквесты не оставались открытыми слишком долго в ожидании ревью, тем самым оптимизируя рабочий поток для потоковых команд. Платформенные команды следят за тем, чтобы максимально возможное количество проверок конфигурации API (например, того, что файлы конфигурации действительны или содержат конкретные настройки) было автоматизировано и интегрировано в пайплайн. Они стремятся оптимизировать общее время исполнения доставки, сокращая время ожидания ревью. Многие платформы хостинга кода предоставляют аналитику статистики пул-реквестов — например, частоту пул-реквестов и продолжительность ревью. Это полезно для отслеживания потока работы в этой части общего процесса создания ценности.

Теперь давайте рассмотрим процесс развертывания конфигурации шлюза в пайплайне из нашего примера.

9.4.5. Развертывание в пайплайне из примера

Конфигурация API-шлюза определяет, как API маршрутизируется вовне, а также политики безопасности, преобразования и оркестрацию, которые могут применяться к запросам и ответам. В рамках APIOps для развертывания конфигурации API поощряется модель GitOps. Она включает следующие шаги.

https://liveinternet.club

1. Команда разработки обновляет декларативную конфигурацию API-шлюза для заданной среды в репозитории контроля версий конфигурации и создает пул-реквест для своих изменений.
2. Обновленные файлы конфигурации подвергаются автоматизированным проверкам. Команда платформы API проводит ревью изменений и, если считает, что все в порядке, одобряет пул-реквест.
3. При слиянии пул-реквеста для согласования и применения желаемой конфигурации к реальному экземпляру шлюза, работающему в среде, запускается программный агент.

Программный агент, применяющий конфигурацию, может быть пайплайном развертывания, который проталкивает конфигурацию в среду шлюза, или оператором Kubernetes в среде, который мониторит Git-репозиторий конфигурации на предмет изменений; обнаружив изменение, он извлекает его и применяет к шлюзу. В следующих разделах я на примерах покажу, как провести согласование с использованием пайплайна развертывания в API-шлюзах Kong и Kusk. В обоих примерах пайплайн включает следующие шаги.

1. Развертывание бэкенд-сервиса API.
2. Согласование конфигурации шлюза.
3. Выполнение тестов соответствия API для проверки того, что конфигурация маршрутизации была успешно применена к API, и для тестирования соответствия API.
4. Если тест соответствия проходит, файл определения API помечается в реестре API как успешно прошедший эту проверку.

Две разновидности репозиториев

Модель GitOps работает с репозиториями кода приложений и репозиториями конфигурации среды. Репозиторий кода приложения хранит исходный код для развернутого бэкенд-приложения. Он также содержит файлы, необходимые для создания и упаковки приложения (например, Docker-файл). Репозиторий конфигурации среды хранит конфигурацию API-шлюза, а также другие конфигурации инфраструктуры и приложений (для развертывания в кластерах Kubernetes они называются манифестами развертывания). Поскольку для этой книги у меня есть только один репозиторий, для хранения кода приложения и конфигурации среды будут использоваться каталоги `chapter8` и `chapter9`.

В разделе 9.4 я представил развертывание сервиса Products API в API-шлюзе в соответствии с принципами APIOps. Сейчас я сделаю то же самое на примере шлюза Kong с учетом описанных ранее концептуальных шагов.

https://liveinternet.club

Развертывание конфигурации в Kong

Сначала включите в файл `build.yaml` код из листинга 9.12, чтобы установить инструмент Kong decK в исполнителе. decK (https://docs.konghq.com/deck) — инструмент для декларативного управления конфигурацией Konig. decK может синхронизировать конфигурацию с кластером Kong и найти расхождения в конфигурации, выявляя дрейф.

Листинг 9.12. Установка decK в исполнителе

```
- name: Deploy-and-Test - Install Kong Deck
  run: |
    curl -sL https://github.com/kong/deck/releases/download/v1.18.0/
➥ deck_1.18.0_linux_amd64.tar.gz -o deck.tar.gz
    tar -xf deck.tar.gz -C /tmp
    sudo cp /tmp/deck /usr/local/bin/
    deck version
```

Затем добавим шаг для развертывания сервиса Products API с Docker Compose. В листинге 9.13 приведен код, который необходимо добавить в файл `.github/workflows/build.yaml`.

Листинг 9.13. Развертывание сервиса Products API

```
- name: Deploy-and-Test - Deploy Products API service
  run: |
    sed -i "s/OWNER/${{ github.repository_owner }}/g"
➥ chapter9/compose-api-service.yaml
    cat chapter9/compose-api-service.yaml
    docker compose -f chapter9/compose-api-service.yaml up -d
```

Добавьте шаг для применения конфигурации маршрутизации для шлюза Kong из файла `chapter9/kong.yaml` в экземпляр шлюза. На этом шаге сначала выполняется проверка файла конфигурации на допустимость. Затем запускается процесс синхронизации для применения желаемого состояния конфигурации в `chapter9/kong.yaml`. Admin API для Kong доступен по адресу http://localhost:8001, и именно с ним взаимодействует команда decK `sync`. Код этого шага приведен в листинге 9.14.

Листинг 9.14. Синхронизация состояния конфигурации Kong

```
- name: Reconcile API gateway state
  run: |
    deck validate --state chapter9/kong.yml
    deck sync --state chapter9/kong.yml --kong-addr http://localhost:8001
```

Теперь добавьте шаги для выполнения интеграционных тестов в скрипт `chapter8/run_integrationtest.sh`. Интеграционные тесты проверяют, что маршрутизация настроена правильно, а API соответствует файлу определения API. Скрипт интеграционного теста вызывает тест `ProductCatalogueEndToEndSchemaTest.java`, который загружает новейший файл определения API из реестра и проверяет его на соответствие.

https://liveinternet.club

Затем добавьте шаг, который использует REST API из реестра API для назначения файлу определения API, прошедшего тест соответствия, метки `passed-conformance-test`. Код этого шага приведен в листинге 9.15.

Листинг 9.15. Выполнение тестов соответствия API

```
- name: Deploy-and-Test - Run API conformance tests
  run: |
    cd chapter8
    ./run_integrationtest.sh
- name: Deploy-and-Test - Mark API definition as passing
➥ API conformance testing
  run: |
    curl -X PUT \
    http://localhost:9090/apis/registry/v2/groups/acmeapis
➥ /artifacts/productsapi/versions
➥ /${{ steps.artifact-version.outputs.ARTIFACT_VERSION }}/meta
➥ --header "Content-Type: application/json"
➥ --data '{"labels": ["passed-conformance-test"] }'
➥ --fail-with-body
```

Создайте пул-реквест и проведите слияние изменений. При слиянии должен запуститься рабочий поток, который завершается неудачей при попытке выполнения интеграционных тестов. Дело в том, что маршрутизация шлюза API для эндпоинта еще не создана.

> **СОВЕТ** Полный файл рабочего потока находится по адресу `.github/workflows/build-kong.yaml.completed` в репозитории исходного кода.

Все готово к созданию конфигурации шлюза. Добавьте конфигурацию для маршрутизации внешнего эндпоинта в бэкенд-сервис API в файл конфигурации Kong `chapter9/kong.yaml`, как показано в листинге 9.16.

Листинг 9.16. Конфигурация маршрутизации для API каталога продуктов

```
_format_version: "2.1"
_transform: true

services:
  - name: Product_Catalog_API
    protocol: http
    host: product-api
    port: 8080
    path:  /
    routes:
      - name: viewProduct
        methods:
          - GET
        paths:
          - "~/v1/catalog/products/(.+)$"
        strip_path: false
```

Маршрутизирует запрос к базовому пути http://product-api:8080/ вышестоящего контейнера docker сервиса Product API

Находит запросы к этому маршруту и отправляет их вышестоящему сервису API, сохраняя путь URL в URL вышестоящего запроса

https://liveinternet.club

Сохраните это изменение и создайте пул-реквест. В этом примере вы сможете одобрить и провести слияние пул-реквеста. Вы увидите, что пайплайн запускается успешно, потому что маршрутизация шлюза теперь существует.

> **СОВЕТ** Полный код примера можно найти в файле `chapter9/build-kong.yaml.completed` в репозитории кода примера.

Итак, мы обновили конфигурацию API-шлюза, чтобы предоставить внешний доступ к эндпоинту сервиса API, и выполнили пайплайн для применения изменений к API-шлюзу с использованием decK. А сейчас я покажу, как сделать нечто похожее с API-шлюзом Kusk, работающим в Kubernetes.

Развертывание конфигурации в Kusk

Kusk — API-шлюз с открытым исходным кодом, который поддерживает внедрение конфигурации API в файлы определения OpenAPI. Он построен на основе высокопроизводительного прокси-сервера Envoy. Kusk развертывается в кластере Kubernetes.

Начнем с добавления маршрутизации Kusk в API каталога продуктов. Для этого добавьте расширение разработчика `x-kusk` для конфигурации маршрутизации в файл определения API `chapter8/apis/product-catalog-api/openapi.yaml`, как показано в листинге 9.17.

Листинг 9.17. Добавление маршрутизации Kusk в определение API

```
…
servers:
  - url: https://api.acme-pet-supplies.co.uk
x-kusk:                                  ← Добавляет расширение x-kusk
  upstream:                                в объект верхнего уровня
    service:
      name: productsapi-svc
      namespace: default
      port: 8080
tags:
  - name: Categories
…
```

Теперь необходимо переписать файл рабочего потока `build.yaml`, чтобы развертывание выполнялось в Kusk, а не в Kong. Удалите из файла `build.yaml` шаги, добавленные для Kong в предыдущем разделе, а затем за шагом `Build - Publish SDK to GitHub Packages` добавьте шаг установки Kusk CLI, как показано в листинге 9.18.

Листинг 9.18. Установка Kusk CLI в пайплайне

```
- name: Deploy-and-Test - Install Kusk CLI
  run: |
     curl -sSLf https://raw.githubusercontent.com/kubeshop
➥ /kusk-gateway/main/cmd/kusk/scripts/install.sh | bash
```

https://liveinternet.club

Шлюз Kusk должен быть установлен в кластере Kubernetes, а значит, последний необходимо создать. Создайте кластер Kubernetes в исполнителе при помощи minikube (https://github.com/kubernetes/minikube) — это простой способ начать работу с Kubernetes для локальной разработки и тестирования. При этом создается кластер Kubernetes, состоящий всего из одного узла, а GitHub Action по адресу https://github.com/medyagh/setup-minikube позволяет включить его в пайплайн.

Добавьте шаги для настройки minikube, установите и настройте Kusk, как показано в листинге 9.19. При этом учтите три обстоятельства.

- При выполнении в пайплайне шага `Install Kusk Gateway` вы можете получить сообщение об ошибке `resource mapping not found for name`[1]`: "kusk-devportal-envoy-fleet"`. В нашем примере это несущественно, не обращайте внимания и переходите далее по варианту `continue-on-error: true`.
- Чтобы отправить запросы шлюзу Kusk в кластере, необходимо настроить переадресацию портов. Это позволяет запросам к локальной машине (исполнителю GitHub) обращаться к ресурсам, выполняемым внутри кластера, — таким, как шлюз Kusk. В данном случае запросы к локальной машине на порте 9000 связываются с портом 80 кластера Kubernetes, где ведет прослушивание шлюз Kusk.
- Чтобы позднее развернуть контейнер сервиса API каталога продуктов в кластере, необходимо предоставить Kubernetes учетные данные (секреты) для получения сервиса из реестра контейнера. Для этого создайте объект секрета Kubernetes с вашими учетными данными GitHub, как показано в листинге 9.19.

Листинг 9.19. Установка шлюза Kusk в пайплайне

```
    - name: Deploy-and-Test - Start minikube
      uses: medyagh/setup-minikube@master
    - name: Deploy-and-Test - Install Kusk Gateway
      continue-on-error: true
      run: |
        kusk cluster install
    - name: Deploy-and-Test - Setup Kusk port forwarding to port 9000
      run: |
        nohup kubectl port-forward svc/kusk-gateway-envoy-fleet
➥ -n kusk-system 9000:80 &
    - name: Deploy-and-Test - Create GitHub Container Registry secret
      run: |
        kubectl create secret docker-registry ghcr-secret
➥ --docker-server=https://ghcr.io
➥ --docker-username=${{ github.repository_owner }}
➥ --docker-password=${{ secrets.GITHUB_TOKEN }}
➥ --docker-email=${{ github.event.pusher.email }}
```

[1] Resource mapping not found for name — Не найдено сопоставление ресурсов для имени. — *Примеч. пер.*

https://liveinternet.club

Затем добавьте шаг для развертывания сервиса `products-api` в кластере. Вам придется заменить некоторые переменные в файле манифеста Kubernetes (`product-api-deployment.yaml`), чтобы Kubernetes смог получить образ из реестра контейнера. В листинге 9.20 показано, как это делается.

Листинг 9.20. Развертывание сервиса Products API в Kubernetes

```
- name: Deploy-and-Test - Deploy products API service
  run: |
    sed -i "s/APP_VERSION/${{ github.run_number }}/g"
➥ chapter9/product-api-deployment.yaml
    sed -i "s/OWNER/${{ github.repository_owner }}/g"
➥ chapter9/product-api-deployment.yaml
          kubectl apply --filename
➥ chapter9/product-api-deployment.yaml
          kubectl apply --filename
➥ chapter9/product-api-service.yaml
```

Заменяет переменную APP_VERSION в файле развертывания

Заменяет переменную OWNER в файле развертывания

Применяет объект развертывания Kubernetes к кластеру

Применяет объект сервиса Kubernetes к кластеру

Затем команда `kusk deploy` используется для развертывания конфигурации API из определения OpenAPI в Kusk. Кроме того, следует проверить статус кластера, чтобы узнать, работают ли поды, как показано в листинге 9.21.

Листинг 9.21. Развертывание конфигурации API в Kusk

```
- name: Reconcile API gateway state
  env:
    LATEST_ARTIFACT_VERSION:
➥ ${{ needs.build.outputs.LATEST_ARTIFACT_VERSION }}
  run: |
    kusk deploy –in
➥ http://localhost:9090/apis/registry/v2/groups/acmeapis/artifacts
➥ /productsapi/versions/$LATEST_ARTIFACT_VERSION
- name: Deploy-and-Test - Check status of the cluster
  run: |
    ./chapter9/check_cluster_status.sh
```

Развертывает определение OpenAPI с конфигурацией API в Kusk

В главе 7 я представил Portman и Newman — инструменты тестирования соответствия. Их можно использовать вместо интеграционных тестов Java, которые мы проводили до этого. Сгенерируйте коллекцию Postman при помощи Portman и выполните коллекцию в Newman. Чтобы сгенерировать удобный отчет HTML с подробной информацией о запросах и ответах, используйте `newman-reporter-htmlextra` (https://mng.bz/lM5j) в качестве репортера Newman. Код приведен в листинге 9.22.

Листинг 9.22. Добавление тестирования соответствия Portman для Kusk

```
- name: Deploy-and-Test - Run API conformance tests
  run: |
```

https://liveinternet.club

```
    cd chapter9
    npm install -g @apideck/portman
    npm install -g newman-reporter-htmlextra
    npm install -g newman
    portman --cliOptionsFile portman/portman-cli.json
    newman run collection.postman.json --verbose
➥ --reporters cli,htmlextra
➥ --reporter-htmlextra-export
➥ ../build-output/conformance-test-report.html
```

Наконец, добавьте шаг для архивации отчета о тестировании (чтобы просмотреть его после выполнения пайплайна) и шаг для пометки файла определения API как прошедшего тестирование соответствия, если выполнение прошло успешно (см. листинг 9.23). Тем самым вы сохраните информацию о том, что определение API прошло тесты соответствия, и будете уверены, что его можно распространять или публиковать на портале разработчика.

Листинг 9.23. Пометка файла API как прошедшего тестирование соответствия в Kusk

```
- name: Deploy-and-Test - Archive conformance test report
  uses: actions/upload-artifact@v3
  with:
    name: conformance-test-report
    path: |
      build-output/conformance-test-report.html
- name: Deploy-and-Test - Mark API definition as
➥ passing API conformance testing
  run: |
    curl --request PUT
➥ http://localhost:9090/apis/registry/v2/groups
➥ /acmeapis/artifacts/productsapi
➥ /versions/${{ steps.artifact-version.outputs.ARTIFACT_VERSION }}/meta
➥    --header "Content-Type: application/json"
➥    --data '{"labels": ["passed-conformance-test-kusk"] }'
➥    --fail-with-body
- name: Mark API definition as passing API conformance testing
  env:
    LATEST_API_VERSION: ${{ needs.build.outputs.LATEST_ARTIFACT_VERSION }}
  run: |
    curl -X PUT http://localhost:9090/apis/registry/v2
➥ /groups/acmeapis/artifacts/productsapi
➥ /versions/$LATEST_API_VERSION/meta
➥ --header "Content-Type: application/json" \
➥ --data '{"labels": ["passed-conformance-test-kusk"] }' \
➥ --fail-with-body
```

И на этом все. Создайте пул-реквест, проведите слияние изменений и понаблюдайте за выполнением рабочего потока.

При запуске завершаются этапы сборки, развертывания и тестирования пайплайна из нашего примера. Итак, мы преобразовали файл определения API,

https://liveinternet.club

сохранили его в реестре API, сгенерировали SDK, развернули конфигурацию API в шлюзе и протестировали API.

Итоги

- На этапе сборки пайплайна развертывания файлы определения API можно обработать и преобразовать для подготовки к внешней публикации. Такое преобразование может включать фильтрацию файла определения API в формате OpenAPI, замену значений, сортировку объектов в файле и т. д.
- Обработанные файлы определений API могут храниться в реестре API, чтобы к ним могли обращаться команды, приложения и системы ниже по потоку. Реестр API служит каталогом API организации; он должен иметь собственный API, который предоставляет программный доступ к артефактам определений API.
- Существуют различные библиотеки и сервисы генерации кода для генерации SDK на основе файлов определений API. Клиентские SDK для API улучшают опыт разработчика для команд, интегрирующихся в API. Пакеты SDK могут храниться в репозитории двоичных артефактов.
- Рабочий поток развертывания APIOps следует практике GitOps по хранению декларативной конфигурации в системе контроля версий. Изменения вносятся с помощью пул-реквестов, а пайплайн развертывания или оператор Kubernetes используется для синхронизации конфигурации с экземпляром API-шлюза.

https://liveinternet.club

10 Снова о соответствии API: специализированный линтинг и проверки безопасности

В этой главе

- ✓ Написание собственных функций линтинга
- ✓ Выполнение проверок линтинга OWASP API
- ✓ Проверка запросов и ответов API в шлюзе
- ✓ Генерация OpenAPI на основе трафика

Чтобы обеспечивать простое и безопасное взаимодействие с разработчиками, интегрирующимися с вашими API, важно продумать автоматизированные методы поддержания соответствия API руководствам по стилю. Заметную роль в этом на стадии проектирования играет линтинг. В главе 3 я рассказал, что это такое и как писать собственные (кастомные) правила линтинга в Spectral с использованием встроенных функций. Впрочем, возможности базовых функций, поддерживаемых инструментом линтинга, ограниченны. Чтобы реализовать правило линтинга на основании руководства по стилю API, необходимо написать собственную функцию, так как встроенных функций может быть недостаточно для требований проекта. В этой главе мы рассмотрим создание собственных функций линтинга.

Другое условие согласованности API, о котором следует помнить, — безопасность. В руководстве по стилю API должны содержаться рекомендации по

https://liveinternet.club

проектированию безопасных API. А если их нет, их необходимо добавить! В этой главе рассматриваются некоторые распространенные угрозы безопасности API и способы проведения проверок линтинга для их обнаружения во время проектирования. Также мы поговорим о том, как заботиться о безопасности на протяжении всего жизненного цикла разработки приложения.

Эта глава написана для разработчиков, и в ней подробно рассматривается тема линтинга API. Исходный код главы доступен в каталоге `chapter10` по адресу https://mng.bz/MZMW.

10.1. Зачем нужны собственные функции

Допустим, вам понадобилось правило линтинга, которое обеспечивает автоматизированную проверку соответствия стандартам, определяемым руководством по стилю API. Представьте, что в руководстве по стилю API вашей организации указано, что имя `operationId` для всех операций GET должно начинаться с `retrieve` или `download`. Допустим, базовые функции вашего инструмента линтинга API не предоставляют нужную для этого функциональность. Как расширить инструмент линтинга для конкретной потребности?

Для этого можно написать собственные (кастомные) правила и функции. Средства линтинга API (такие, как Spectral, Redocly CLI и Vacuum) предоставляют возможность расширения функциональности при помощи собственных правил линтинга, которые указывают, где в документе определения API должна происходить проверка линтинга и какие кастомные функции вызывают кастомный код для проверки. В главе 3 было показано, как создавать собственные правила. Чтобы применить их в данном случае, следует сначала проверить, существует ли в используемом инструменте линтинга API встроенное правило, с помощью которого можно получить эту функциональность, или общедоступный набор правил, который можно было бы импортировать. Если проверка ничего не даст, придется написать правило самостоятельно. В таком случае рекомендуется начать с проваливающегося юнит-теста, отражающего нужное поведение. Затем можно написать собственное правило и функцию для нужной проверки. Мы поговорим об этом позже.

10.1.1. Поиск повторно используемых правил

Написание собственной функции для собственного правила означает код, который придется обслуживать. А значит, прежде чем его писать, следует проверить, нет ли в используемом инструменте линтинга встроенного правила, которое можно было бы просто активировать и изменить его настройки критичности. Например, `operation-operationId-unique` — встроенное правило Spectral (то есть `spectral:oas`), по умолчанию имеет уровень критичности `warning`. Ему можно назначить уровень критичности `error`:

https://liveinternet.club

```
extends: spectral:oas
rules:
  oas3-unused-component: error
```

При использовании инструмента линтинга ознакомьтесь с его встроенными правилами. Ниже перечислены некоторые источники, в которых можно найти встроенные наборы правил популярных инструментов линтинга.

- Для Spectral список правил встроенного набора OpenAPI доступен по адресу https://mng.bz/Bdrv. Как говорилось в главе 3, для включения этого набора правил следует добавить элемент `extends: spectral:oas` в файл `spectral.yaml`. При этом активируется минимальный набор правил (то есть правил с пометкой `Recommended` в документации). Чтобы активировать все правила, следует добавить элемент `[[extends: spectral:oas, all]]`.
- Redocly CLI предоставляет минимальный набор с правилами, список которых доступен по адресу https://redocly.com/docs/cli/rules/minimal, и расширенный рекомендуемый набор правил по адресу https://redocly.com/docs/cli/rules/recommended.
- Vacuum также предоставляет минимальный рекомендуемый набор правил (https://quobix.com/vacuum/rulesets/recommended) и расширенный набор, содержащий все правила (https://quobix.com/vacuum/rulesets/all).

Если во встроенном наборе нет правила, нужного вам, рассмотрите возможность импортирования доступных наборов, в которых оно есть. В главе 3 (раздел 3.8) я назвал общедоступные наборы правил, которые можно импортировать для Spectral. На странице https://github.com/stoplightio/spectral-rulesets также размещается список таких общедоступных наборов, поддерживаемый сообществом Spectral.

Если вам не удалось найти набор с открытым исходным кодом, содержащий нужное правило, можно написать собственное, которое использует базовую функцию инструмента линтинга API. Если такая функция не найдется, придется писать собственную функцию. Это вариант на крайний случай. Например, представьте, что в описанном выше сценарии вы используете Spectral. Чтобы избежать написания собственной функции, можно воспользоваться одной из базовых функций Spectral — функцией `pattern`, которая сравнивает целевое значение с регулярным выражением, и определить правило `operationId-get-should-follow-naming-convention`.

```
rules:
  operationId-get-should-follow-naming-convention:
    description: 'GET operations should have an operationId
➥ starting with retrieve or download'[1]
```

[1] Текст описания: ‘operationId для всех операций GET должно начинаться с retrieve или download’ (в коде не рекомендуется использовать кириллические символы, лучше писать на английском). — *Примеч. науч. ред.*

https://liveinternet.club

```
message: "GET operationId should follow naming convention. {{error}}"[1]
given: "$.paths[*]['get']['operationId']"
severity: error
then:
  function: pattern
  functionOptions:
    match: "(^retrieve\\w*)|(^download\\w*)"
```

Использует встроенную функцию pattern в Spectral

Предоставляет регулярное выражение для проверки

А теперь я покажу, как реализовать нестандартную функцию для этого сценария, чтобы вы представляли, как это делается.

10.1.2. Написание теста для правила линтинга

Как уже говорилось, прежде чем писать новое собственное правило, желательно создать для него набор тестов. В этом случае можно использовать метод разработки через тестирование. Тесты обеспечат защиту, если в будущем потребуется провести рефакторинг правила.

Здесь действуют все обычные принципы написания юнит-тестов. Тест должен ясно показать, что именно вы тестируете. Он должен вызвать тестируемую операцию (например, имя правила) с данными, на которых она тестируется, — то есть содержать пример определения API, для которого правило должно либо пройти, либо завершиться с сообщением об ошибке. Определением API не обязательно должен быть большой файл OpenAPI. Это может быть фрагмент, содержащий важные части; необходимо лишь убедиться, что намерение теста выполняется. Также желательно через оснастку тестов отключить все остальные правила, кроме тестируемого. В этом случае правило линтинга вернет только интересующие вас сообщения об ошибках. Наконец, тест должен проверять сообщения об ошибках, возвращаемые инструментом линтинга (если таковые имеются). Обычно отсутствие сообщений означает, что определение API прошло линтинг.

Существуют паттерны, используемые различными инструментами линтинга для проверки правил. Первый паттерн — включение как тестируемого определения API, так и сообщений об ошибках для него в одну тестовую функцию. Другой — хранение тестируемого определения API и ожидаемого выходного сообщения линтинга в файлах отдельно от самого теста. Если вы используете линтер с открытым исходным кодом, я рекомендую просмотреть документацию или репозиторий проекта, чтобы узнать, какие паттерны используются для тестирования правил линтинга.

Вернемся к сценарию, с которого я начал. Напишем тесты счастливого пути (happy path test) и несчастливого пути (sad path test) нового правила. Структура этих тестов изображена на рис. 10.1.

---

[1] Текст сообщения об ошибке: «Для operationId операций GET должно соблюдаться соглашение об именовании». — *Примеч. науч. ред.*

https://liveinternet.club

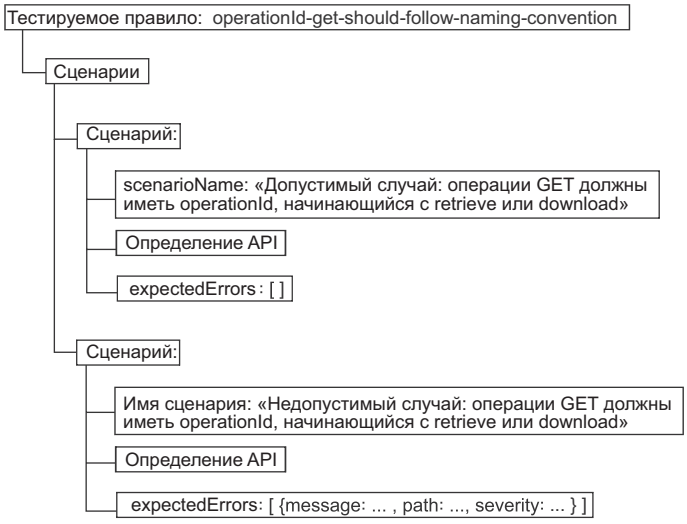


Рис. 10.1. Структура теста правила линтинга API для проверки префикса operationsId

Структура юнит-теста основана на паттерне из проекта Spectral. Это позволяет разработчикам расширять тесты, не обладая глубокими знаниями используемого языка (JavaScript или TypeScript), так как в тесте обязательны только три поля — имя сценария, пример определения API и ожидаемые сообщения о проблемах. Если вы используете другой инструмент линтинга с открытым исходным кодом, поищите в его коде паттерны, используемые для тестирования правил.

Теперь напишите код для реализации теста. Но сначала перейдите в каталог `chapter10` и выполните команду `npm install` для установки всех необходимых пакетов NPM. Затем напишите сценарий счастливого пути, который не возвращает сообщений об ошибках, если определение API не нарушает правила линтинга. Код приведен в листинге 10.1; вы можете сохранить его в файле `chapter10/__tests__/operation-id-pattern.test.ts` при помощи Visual Studio Code (VS Code) или предпочитаемой IDE.

Листинг 10.1. Юнит-тесты для собственного правила линтинга

```
import { DiagnosticSeverity } from '@stoplight/types';
import testRule from '../__helpers__/helper';

testRule(  'operationId-get-should-follow-naming-convention', [
➥ {
        scenarioName: ' Valid case: GET operations should have an
➥ operationId starting with retrieve or download'[1],
```

Импортирует вспомогательную функцию, содержащую функции DSL

[1] Имя сценария: «Допустимый случай: операции GET должны иметь operationId, начинающийся с retrieve или download». — *Примеч. пер.*

https://liveinternet.club

```
        apiDefinition: {                          ← Создает определение API
            openapi: '3.1.0',                       в виде объекта JavaScript
            info: { version: '1.0' },
            paths: {
                '/v1/users': {
                    get: {
                        operationId: 'retrieveUsers',   ← Следует предполагаемым
                    }                                     соглашениям
                }
            }
        },
        expectedErrors: [],   ← В этом сценарии ошибки
    },                          не ожидаются
```

Функция `testRule()` активирует линтер Spectral и отключает все правила, кроме тестируемого. Затем расширьте тесты и добавьте сценарий для несчастливого пути. Проверьте, что при передаче определению API значения `operationId`, не соответствующего соглашению, возвращается правильное сообщение об ошибке, как показано в листинге 10.2.

Листинг 10.2. Сценарий несчастливого пути в юнит-тесте для правила линтинга

```
…
{
        scenarioName: ' Invalid case: GET operations should have an
➥ operationId starting with retrieve or download'[1],
        apiDefinition: {
            openapi: '3.1.0',
            info: { version: '1.0' },
            paths: {
                '/v1/users': {
                    get: {
                        operationId: 'listUsers',   ← Имя OperationId не
                    }                                 соответствует соглашению
                }
            }
        },
        expectedErrors: [   ← Массив ожидаемых объектов
            {                   сообщений об ошибках
                message: "GET operationId should follow naming convention.
➥OperationId 'listUsers' should start with 'retrieve' or 'download'.",
                path: ['paths', '/v1/users', 'get', 'operationId'],
                severity: DiagnosticSeverity.Error,
            }
        ],
    }
]);
```

[1] Имя сценария: «Недопустимый случай: операции GET должны иметь operationId, начинающийся с retrieve или download». — *Примеч. пер.*

https://liveinternet.club

Теперь запустите тест, выполнив команду `npm test` в командной строке. Тест должен провалиться с сообщением об ошибке `ReferenceError: Cannot extend non-existing rule`[1]`: "operationId-get-should-follow-naming-convention`, потому что собственное правило еще не написано. Пока все в порядке. Теперь напишем это правило.

10.1.3. Написание собственного правила

Прежде всего необходимо задать описание собственного правила, сообщение, выражение `given` и функцию. Написание собственных правил Spectral рассматривалось в главе 3. Напомню, что выражение `given` определяет часть файла определения API, к которому должна применяться проверка. Это должно быть выражение JSONPath, которое в данном случае описывает все `operationId` для операций GET в файле OpenAPI. Сценарий отличается тем, что правило должно вызывать собственную функцию, которую вы еще не определили. Допустим, вы хотите назвать собственную функцию `getOperationIdConvention`. На рис. 10.2 изображена структура собственного правила линтинга.

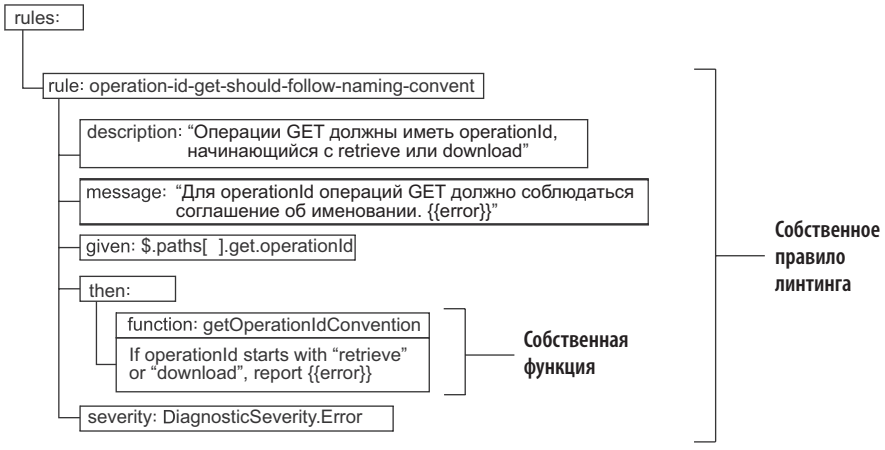


Рис. 10.2. Структура собственного правила линтинга Spectral для проверки operationId операций GET

На этот раз запишите правило в файл `ruleset.ts`, определяя новое правило `operationId-get-should-follow-naming-convention` на TypeScript (а не на YAML, как делалось в главе 2), см. листинг 10.3.

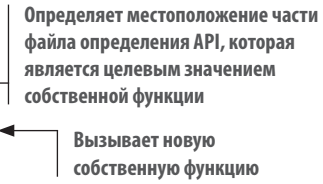
Листинг 10.3. Собственное правило для проверки префикса operationId

```
export default {
    rules: {
```

[1] Невозможно расширить несуществующее правило. — *Примеч. пер.*

https://liveinternet.club

```
        'operationId-get-should-follow-naming-convention': {
            description: 'GET operations should have an operationId
➥ starting with retrieve or download',
            message: "GET operationId should follow
➥ naming convention. {{error}}",
            given: "$.paths[*].get.operationId",
            then: {
                function: getOperationIdConvention
            },
            severity: DiagnosticSeverity.Error,
        },
```



Следующий шаг – определение собственной функции `getOperationIdConvention`.

10.1.4. Определение собственной функции

В популярных инструментах линтинга — таких, как Spectral, Redocly и Vacuum, — собственные функции линтинга получают целевой объект (я буду называть его «целевым значением») в качестве входных данных из определения API и возвращают сообщение, если цель не проходит проверку, реализованную в функции. При написании собственных функций соблюдайте все рекомендации по удобочитаемости кода. Например, разбивайте большие функции на несколько меньших, каждая из которых будет наделена единственной обязанностью. Для удобства также полезно оставлять комментарии в коде или использовать языки с сильной типизацией – такие, как TypeScript. TypeScript основан на JavaScript, но обеспечивает лучшую типобезопасность.

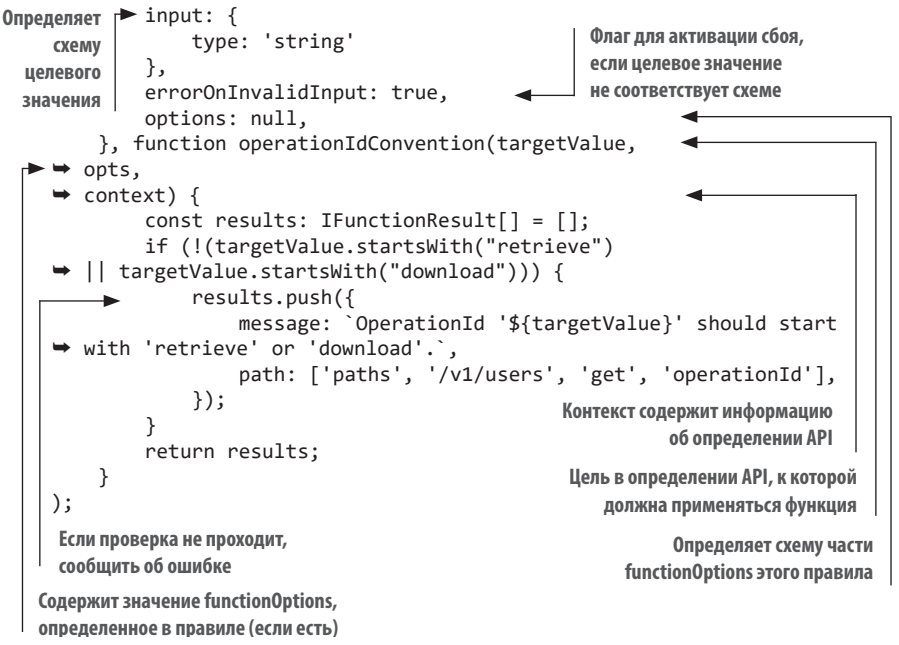
Посмотрим, как реализовать собственную функцию `getOperationIdConvention`. Она должна проверить входящее целевое значение — `operationId`, найденное условием `given` правила линтинга, и, если оно не начинается с `retrieve` или `download`, сообщить об ошибке в файле определений. Это может быть функция TypeScript, которая принимает целевое значение (определяемое по условию `given`, найденному с помощью собственного правила), ожидаемая схема целевого значения, значение переданных параметров функции и объект контекста, содержащий более подробную информацию о файле определения API и наборе правил. Она возвращает массив, содержащий сообщение об ошибке линтинга (если есть). Определите новую собственную функцию `getOperationIdConvention` в файле `chapter10/functions/get-operation-id-naming.ts`, как показано в листинге 10.4.

Листинг 10.4. Определение собственной функции в TypeScript

```
import {createRulesetFunction} from "@stoplight/spectral-core"
import type {IFunctionResult, RulesetFunctionContext}

➥ from '@stoplight/spectral-core';
   type Input = string;
   type Options = null;
   export default createRulesetFunction<Input, Options>({
```

https://liveinternet.club




```
input: {
    type: 'string'
},
errorOnInvalidInput: true,
options: null,
}, function operationIdConvention(targetValue,
➥ opts,
➥ context) {
    const results: IFunctionResult[] = [];
    if (!(targetValue.startsWith("retrieve")
➥ || targetValue.startsWith("download"))) {
        results.push({
            message: `OperationId '${targetValue}' should start
➥ with 'retrieve' or 'download'.`,
            path: ['paths', '/v1/users', 'get', 'operationId'],
        });
    }
    return results;
}
);
```

Теперь запустите тест. Должно появиться сообщение о его прохождении:

```
$>: npm test
> apiopsbook@1.0.0 test
> jest
 PASS  __tests__/operation-id-pattern.test.ts
  Rule operationId-get-should-follow-naming-convention
    ✓ Valid case: GET operations should have an operationId
➥ starting with retrieve or download (1 ms)
    ✓ Invalid case: GET operations should have an operationId
➥ starting with retrieve or download
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.942 s, estimated 3 s
Ran all test suites.
```

Все тесты проходят, а реализация завершена. Помните, что собственную функцию также можно записать в виде более простой функции JavaScript. Например, ее можно сохранить в файле `chapter10/functions/get-operation-id-naming-simple.js`:

```
module.exports = function getOperationIdConvention(targetValue) {
    const results = [];
    if (typeof targetValue !== 'string'){
        results.push({message: `"operationId" must be a string`});
     }
    if (!(targetValue.startsWith("retrieve") ||
➥ targetValue.startsWith("download"))) {
        results.push({
```

Проверяет тип для предотвращения ошибки TypeError

https://liveinternet.club

```
            message: `OperationId '${targetValue}' should start with
➥ 'retrieve' or 'download'.`,
            path: ['paths', '/v1/users', 'get', 'operationId'],
        });
    }
    return results;
}
```

Чтобы использовать этот код, необходимо импортировать его в файл `ruleset.ts` следующией директивой:

```
import getOperationIdConvention from
➥ './functions/get-operation-id-naming-simple';
```

Вы успешно написали свою функцию и выполнили тесты. Теперь перечислим рекомендации, которых необходимо придерживаться при написании собственных функций.

10.1.5. Что необходимо помнить при написании собственных функций

Если вы работаете в большой команде, предоставляйте документацию о том, как добавлять новые правила, изменять существующие или игнорировать их там, где это необходимо. Документация также должна объяснять, как добавлять новые юнит-тесты для собственных правил. Это поможет разработчикам действовать самостоятельно, добавляя новые правила через пул-реквесты. Конечно, при игнорировании правил необходима осторожность, и член команды, отправляющий такое изменение, должен объяснить, почему правило необходимо игнорировать (например, в сообщении пул-реквеста).

При введении нового собственного правила следует помнить, что его не обязательно включать для всех API сразу, поскольку это может привести к ошибке линтинга для существующих API, не проходящих проверку. Правило можно вводить в API постепенно, выбирая, какие наборы правил добавлять и какие правила активировать.

Линтинг может выполняться по-разному — например, в редакторе кода, CLI или в браузере. Поэтому собственные функции должны быть детерминированными и в общем случае избегать побочных эффектов. Например, следует тщательно продумать, использовать ли побочные эффекты (скажем, логику кэширования).

Кроме того, необходимо тщательно продумать последствия от применения внешних библиотек или API, которые могут иметь проблемы кросс-платформенной совместимости. Например, использование встроенного API проверки орфографии в операционной системе Windows может ограничить портируемость собственной функции. Другой пример — выдача HTTP-запросов; для них придется учитывать любые требования аутентификации и другие потенциальные

https://liveinternet.club

трудности — такие, как ограничения совместного использования ресурсов между различными источниками (CORS, Cross-Origin Resource Sharing) в браузере.

Я показал, как определять собственные функции. Теперь обсудим важную область, в которой вам, возможно, придется применять эти знания.

10.2. Автоматизированное ревью безопасности определений API

Представьте, что ваша компания работает над созданием API и руководитель просит вас провести ревью предложения API и убедиться, что оно безопасно. Файл определения OpenAPI, ревью которого надо провести, находится в репозитории исходного кода `chapter10/crapi.api.yaml`. Как проверить, что API безопасен и не содержит известных уязвимостей?

Чтобы обнаружить уязвимости, необходимо иметь базовое представление о безопасности API. Начать изучение вопросов безопасности стоит с файла `crapi.api.yaml`, так как он намеренно проектировался небезопасным. Я скопировал его из проекта Completely Ridiculous API[1] (crAPI) организации OWASP (Open Worldwide Application Security Project) https://github.com/OWASP/crAPI. Приложение crAPI используется для заказа техобслуживания автомобилей. Пользователи могут создавать учетные записи, искать запчасти, отправлять запросы на сервис и приобретать аксессуары. Также у приложения есть форум, на котором пользователи могут публиковать комментарии и посты в блогах. Сервисы, предоставляемые через API проекта crAPI, показаны на рис. 10.3.

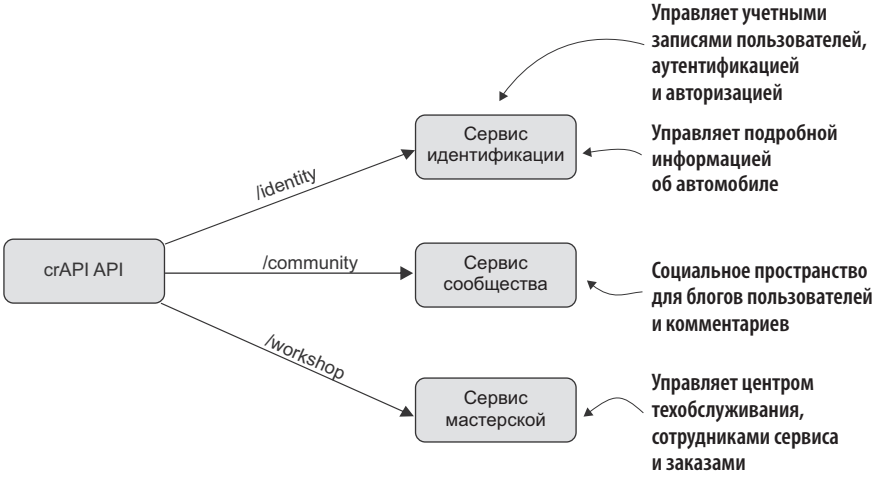


Рис. 10.3. Микросервисы бэкенда, формирующие crAPI

[1] «Совершенно нелепый API». — *Примеч. пер.*

https://liveinternet.club

Если вам интересно, вы найдете полную реализацию сервисов crAPI в репозитории GitHub проекта. Впрочем, в этой главе мы прежде всего займемся проверкой дизайна API на наличие дефектов безопасности. Однако начнем с краткого введения в безопасность API.

10.2.1. Введение в безопасность API

Безопасность API подразумевает устранение рисков для бизнес-ресурсов, с которыми работает API. Она включает защиту API, предоставляемых потребителям, с использованием таких механизмов, как шифрование, аутентификация, авторизация, журналы аудита, ограничение частоты обращений и т. д. Безопасность API — обширная область, охватывающая безопасность информации, безопасность сети и безопасность приложений. Проверки безопасности для защиты от уязвимостей API необходимо проводить на каждом этапе его жизненного цикла — проектирования, разработки, тестирования и эксплуатации. *Уязвимостями API* называются нарушения, дефекты или недостатки API, которые могут эксплуатироваться злоумышленниками для ослабления защиты системы.

Так как API являются интерфейсами к программным системам, важно учитывать актуальные для этих систем угрозы безопасности. Суть *моделирования угроз* заключается в выявлении и понимании потенциальных угроз для программных систем, вероятности их возникновения, возможных последствий, ключевых мер управления безопасностью и контрмер, которые должны быть приняты для защиты системы. Вероятность и последствия проблем важны для понимания риска, поскольку маловероятная проблема с серьезными последствиями может оказаться опаснее, чем вероятная проблема с незначительными последствиями.

Термины, используемые при моделировании угроз

- *Угроза* — потенциальное событие, которое может повредить или нарушить работоспособность бизнес-ресурса или бизнес-цели.
- *Уязвимость* — дефект в реализации системы, который может быть использован злоумышленником.
- *Атака* — исполнение угрозы: попытка похищения, раскрытия, модификации, уничтожения, блокировки или получения несанкционированного доступа к системе или бизнес-ресурсу либо их использования.
- *Контрмеры* — действия, предпринимаемые по отношению к уязвимостям для снижения вероятности атак и последствий угроз.
- *Вектор атаки* — канал, по которому проводятся атаки (например, клиентский браузер или сетевой прокси-сервер).
- *Актор* — человек или группа, способные проводить функциональное использование или злоупотребление системой.
- *Последствия* — ущерб от атаки.

https://liveinternet.club

Как утверждается в книге «Threat Modeling: A Practical Guide for Development Teams» Изара Тарандача (Izar Tarandach) и Мэтью Коулза (Matthew Coles) (O'Reilly, 2020), любой метод моделирования угроз по сути основан на ответах на четыре вопроса.

- *Что вы создаете*? Смоделируйте систему, которую вы создаете, с помощью диаграмм потоков данных, моделей сценариев использования и архитектурных диаграмм, чтобы анализ безопасности выполнялся более эффективно.
- *Что может пойти не так*? Используйте структурные методы и *деревья атаки* (диаграммы, показывающие разные способы вмешательства в работу системы) для исследования и выявления частей системы, которые могут подвергнуться атаке. Пример структурированного подхода к поиску угроз — метод STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege — спуфинг, фальсификация, отказ, раскрытие информации, отказ в обслуживании, повышение привилегий).
- *Что с этим делать*? Предложите меры безопасности или средства контроля для устранения выявленных угроз, атак или рисков. Выявленные риски должны стать приоритетными, а предложенные средства контроля переданы в процесс проектирования и разработки и пайплайн CI/CD, который автоматизирует контроль, чтобы полученный продукт был безопасным.
- *Насколько хорошо вы справились с задачей*? Проверьте, что угрозы устранены. Непрерывно совершенствуйте процесс моделирования угроз, регулярно проводя ревью и проверяя, не было ли что-то упущено и можно ли что-то улучшить.

На рис. 10.4 показаны эти шаги. Более масштабное обсуждение инструментов и методологий моделирования угроз выходит за рамки книги, но вы найдете их обзор в статье OWASP «Threat Modeling Cheat Sheet» на странице https://mng.bz/aEl7. Вопросы безопасности API подробно раскрываются в книге «API Security in Action» Нила Мэддена (Neil Madden) (Manning, 2020; www.manning.com/books/api-security-in-action).

OWASP API Security Top 10 (https://mng.bz/gvlV) — документ, составленный отраслевыми экспертами и представляющий общий обзор критических угроз безопасности, которые затрагивают веб-API. На рис. 10.5 показаны угрозы, перечисленные в версии 2023 года.

В следующем разделе я кратко представлю содержание OWASP API Security Top 10. Мы не будем подробно обсуждать, как устранить эти уязвимости, — информацию об этом можно найти на сайте OWASP: https://mng.bz/gvlV. Но я покажу, как выявлять признаки следующих пяти уязвимостей посредством линтинга файла определения API:

- API1:2023 — нарушения авторизации на уровне объектов;
- API2:2023 — нарушения аутентификации;
- API3:2023 — нарушения авторизации на уровне свойств объектов;

https://liveinternet.club

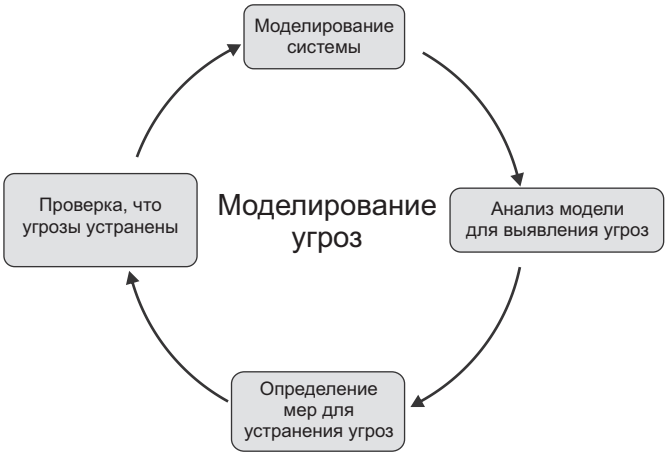


Рис. 10.4. Основные этапы моделирования угроз

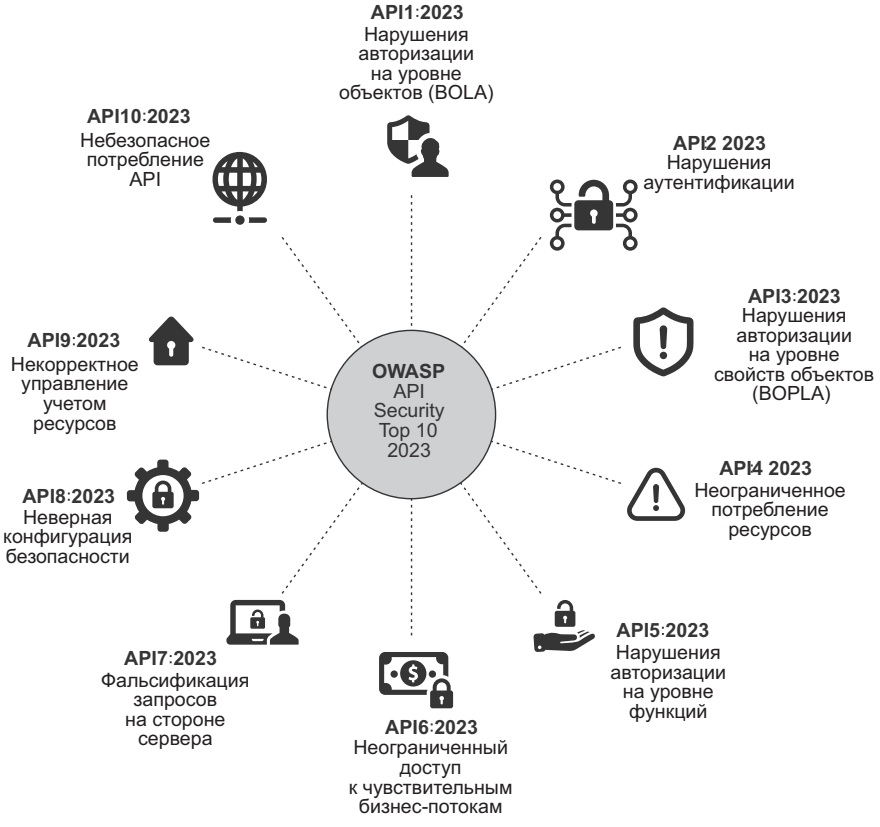


Рис. 10.5. Первая десятка рисков безопасности API, выявленная OWASP

https://liveinternet.club

- API4:2023 — неограниченное потребление ресурсов;
- API8:2023 — неправильная конфигурация безопасности.

Начнем с краткого знакомства со списком API Security Top 10.

10.2.2. API1:2023. Нарушения авторизации на уровне объектов

Эндпоинты API могут предоставлять идентификаторы объектов ресурсов — например, объект `accountId` в `GET /accounts/{accountId}`. Уязвимость BOLA (Broken Object Level Authorization) встречается там, где злоумышленник может изменить идентификатор объекта, отправленный в запрос; это позволит ему получить доступ к ресурсу, который должен быть для него недоступным. В нашем примере вместо своего `accountId` он присылает `accountId` другого пользователя. Это может привести к несанкционированному доступу, изменению информации или ее уничтожению.

Один из способов получения злоумышленником идентификатора объекта основан на перехвате трафика API от других пользователей. Кроме того, злоумышленник может попытаться подобрать идентификатор. Подбор упрощается эндпоинтами, которые используют числовые, последовательные идентификаторы объектов вида `GET /accounts/123` и `GET /accounts/124`. Таким образом, использование предсказуемых идентификаторов объектов считается плохой практикой. Это один из признаков рассматриваемой уязвимости, который можно искать в файлах определений API. И конечно, в этом вам поможет линтинг API. Опробуем эту возможность на файле `crapi.api.yaml` в следующем упражнении.

Я создал в каталоге `chapter10` набор правил линтинга (`chapter10/.spectral.yaml`), объединяющий набор правил Spectral OWASP (https://mng.bz/eoAQ), который на момент написания книги основывался на рисках OWASP 2019, с некоторыми кастомными правилами линтинга, также относящимися к OWASP (о них чуть позже).

> **ПРИМЕЧАНИЕ** Между рисками OWASP 2023 и OWASP 2019 существует ряд различий. В тех случаях, когда эти различия влияют на набор правил Spectral OWASP 2019, использованный в этом упражнении, я буду на это указывать.

Запустите набор правил линтинга из файла `crapi.api.yaml` следующими командами:

```
cd chapter10
npm install
spectral lint crapi.api.yaml  > issues.log
```

Устанавливает зависимости, определенные в package.json

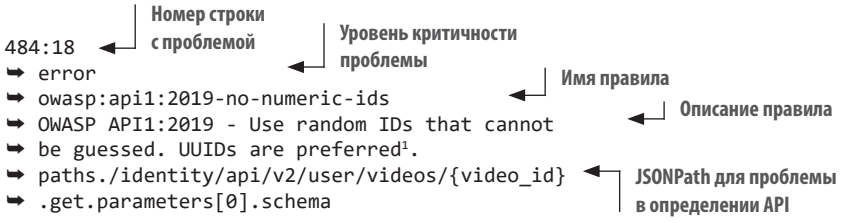
Проводит линтинг и записывает результаты в файл

Теперь найдите в файле сообщения о проблемах, обнаруженных с помощью линтинга, которые начинаются с `'owasp:api1'`; для этого выполните команду:

```
grep -E 'owasp:api1' issues.log
```

https://liveinternet.club

Команда выводит несколько сообщений:



Из результатов видно, что некоторые эндпоинты используют числовые идентификаторы, которые злоумышленник может подобрать. Чтобы избежать этого риска, в дизайне эндпоинтов следует использовать UUID (Universally Unique Identifier). В табл. 10.1 описана эта уязвимость и то, как ее обнаружить с помощью линтинга.

Таблица 10.1. Обнаружение BOLA с помощью линтинга

Уязвимость	Возможный эксплойт	Обнаружение с помощью линтинга API
Операции API содержат идентификаторы объектов ресурсов, которые могут быть подобраны злоумышленником	Злоумышленник может угадать идентификатор объекта или обнаружить идентификатор, проведя мониторинг трафика запросов. Он может воспользоваться этой возможностью для запроса объекта, который должен быть для него недоступен	Включите правила линтинга, которые проверяют, что идентификаторы важных объектов в путях URL, заголовках, телах запросов и ответов используют случайные идентификаторы, которые злоумышленник не сможет подобрать (например, UUID)

Стоит подчеркнуть, что такие проверки линтинга не смогут обнаружить все экземпляры уязвимости BOLA. Например, злоумышленник может использовать эндпоинт `GET /community/api/v2/community/posts/recent` для получения списка постов, включающий авторов и связанные с ними идентификаторы автомобилей (UUID, которые невозможно подобрать), а затем использовать эти идентификаторы для получения местонахождения автомобиля через эндпоинт `GET /identity/api/v2/vehicle/{vehicleId}/location`. Если для эндпоинта не предусмотрены должные проверки авторизации (как в случае реализации этого намеренно уязвимого приложения), запрос вернет конфиденциальную информацию о местонахождении автомобиля.

[1] Use random IDs that cannot be guessed. UUIDs are preferred — Используйте случайные идентификаторы, которые невозможно подобрать. Использование UUID предпочтительно. — *Примеч. пер.*

https://liveinternet.club

10.2.3. API2:2023. Нарушения аутентификации

Нарушение аутентификации встречается в плохо реализованных потоках аутентификации. Многие API имеют специальные эндпоинты для аутентификации и регистрации пользователей в системе. Эти эндпоинты устанавливают *механизм безопасности* для обращения к другим эндпоинтам в API. Потоки аутентификации обычно сложны, и не каждый инженер, работающий в системе, сможет понять, как они работают или как правильно их реализовать. Из-за этого эндпоинты аутентификации обычно служат целью специальных атак, поскольку злоумышленник может использовать плохо реализованные потоки аутентификации, чтобы выдать себя за другого пользователя, взять под контроль его учетную запись и выполнять опасные действия от его лица.

Механизмы безопасности OpenAPI

Механизм безопасности представляет собой метод или процедуру, необходимую, чтобы выполнить требования безопасности и обеспечить соблюдение политики безопасности. Для API это определенный метод аутентификации и авторизации запросов к API, и для передачи этой информации потребителям используется определение API. В OpenAPI для обращения к механизмам аутентификации и авторизации, доступным API, применяется термин *«схема безопасности»*.

Спецификация OpenAPI использует *объект схемы безопасности*, чтобы у проектировщиков API была возможность определить необходимые механизмы аутентификации и авторизации. OpenAPI 3.1 поддерживает следующие схемы безопасности.

- *HTTP Authentication*. Использует заголовок HTTP Authorization, определенный в RFC 7235. HTTP Authentication поддерживает разные виды схем аутентификации, включая Basic, Bearer, Digest и др. (полный список см. в реестре схем аутентификации IANA по адресу https://mng.bz/5lEB). Например, при базовой аутентификации пользователь отправляет учетные данные в виде пар «идентификатор пользователя — пароль», закодированные в формате Base64: `Authorization: Basic bXl1c2VyaWQ6IG15cGFzc3dvcmQ==`. Сам по себе этот метод не считается безопасным, если только он не объединяется с механизмом шифрования транспортного протокола, по которому передаются учетные данные.
- *API Keys*. Ключ API представляет собой токен (предоставляемый потребителю API через некоторый внешний процесс), отправляемый серверу API. Пользователь отправляет его серверу (в заголовке API, cookie или параметре запроса) для выявления и аутентификации вызывающего клиентского приложения. Отправлять ключ API в параметре запроса не рекомендуется, так как при этом возникает риск перехвата и использования ключа API злоумышленником.
- *Mutual TLS* (mTLS). При использовании mTLS клиент и сервер проводят аутентификацию друг друга с помощью сертификатов X.509, часто используемых в B2B-приложениях с повышенными требованиями к безопасности. mTLS отличается от обычного протокола TLS (Transport Layer Security), который требует только чтобы сервер подтвердил свои данные клиенту с использованием сертификата X.509.
- *OAuth2*. Это фреймворк авторизации, который позволяет клиентскому приложению обратиться к защищенному ресурсу с согласия владельца ресурса. Владелец

https://liveinternet.club

ресурса предоставляет доступ к клиентскому приложению, а клиентское приложение обменивает заявку на токен доступа. Токен содержит информацию об области действия, продолжительности и других атрибутах заявки, а клиентское приложение предоставляет эту информацию серверу ресурса при совершении запроса.

OIDC (OpenID Connect). OpenID Connect обеспечивает уровень идентификации на базе протокола OAuth2, позволяющего клиентским приложениям проверять идентичность конечных пользователей и получать информацию об их профилях. Он определяет поток, в котором клиентское приложение может получать *утверждения* (claims) — фрагменты информации об аутентифицированном пользователе, например, имя или адрес электронной почты, от *сервера авторизации* (провайдера идентификации) в виде токена идентификации JWT (JSON Web Token).

Теперь проверьте эту уязвимость в файле `crapi.api.yaml`, выполнив следующую команду:

```
grep -E 'owasp:api2' issues.log
```

Команда возвращает информацию о нескольких проблемах:



Результаты линтинга показывают, что в файле определения имеются незащищенные эндпоинты. `owasp:api2:2019-protection-global-rule` сообщает о том, что существуют эндпоинты, не имеющие схем безопасности, определенных на глобальном или операционном уровне. Но у общедоступных эндпоинтов аутентификации, относящихся к регистрации и входу, схема безопасности определяться не должна, чтобы пользователи могли использовать их для входа. Это пример того, что для этих эндпоинтов аутентификации в правилах линтинга должно присутствовать переопределение (override) с игнорированием правила `owasp:api2:2019-protection-global-unsafe`. Но стоит еще раз подчеркнуть: это не означает, что в API отсутствуют уязвимости нарушений аутентификации. Если злоумышленнику известен адрес электронной почты пользователя, он может воспользоваться функциональностью смены пароля и методом брутфорса, чтобы совершить запросы к `POST /identity/api/auth/v2/check-otp` с целью

[1] This operation is not protected by any security scheme — Данная операция не защищена ни одной схемой безопасности. — *Примеч. пер.*

https://liveinternet.club

найти действительный одноразовый пароль для сброса пароля пользователя. Впрочем, обсуждение этой темы выходит за рамки книги.

10.2.4. API3:2023. Нарушения авторизации на уровне свойств объектов

Объекты ресурсов, возвращаемые API, могут содержать несколько полей — то есть *свойств* объекта. Некоторые API возвращают все свойства объекта. Уязвимость BOPLA (Broken Object Property Level Authorization), то есть нарушения авторизации на уровне свойств объектов, возникает, когда некоторые из возвращаемых свойств содержат конфиденциальную информацию, которая должна быть недоступной для пользователя. Так как для возвращаемых полей не выполняются нужные проверки свойств, это может привести к тому, что пользователь сможет изменить чувствительные свойства. Несанкционированное раскрытие данных и подмена данных в результате этой уязвимости могут привести к тому, что злоумышленник сможет взять под контроль учетные записи пользователей.

Теперь проверьте этот риск в файле `crapi.api.yaml`, выполнив команду `grep`, которая ищет проблемы Spectral для имен, начинающихся на `owasp:api3:` или `owasp:api6:`:

```
grep -E 'owasp:api3:|owasp:api6:' issues.log
```

Заметим, что выражение `grep` в этой команде выполняет поиск проблем, связанных с правилами, начинающимися с `owasp:api3:` и `owasp:api6:`, поскольку риск 3 для API (нарушения авторизации на уровне свойств объектов) в OWASP Top 10 версии 2023 года является комбинацией риска 3 (избыточное раскрытие данных — когда эндпоинт раскрывает конфиденциальные поля, не предназначенные для чтения пользователем) и риска 6 (массовое присваивание – когда эндпоинт предоставляет пользователю несанкционированный доступ для изменения или удаления конфиденциального поля) версии 2019 года. Напомню, что набор правил, которые я здесь описываю, импортирует набор правил Spectral OWASP 2019 года.

Команда возвращает ошибки следующего вида:

```
862:20
➥ warning
➥ owasp:api3:2023-no-default-additionalProperties
➥ Objects should not allow additionalProperties.
➥ Disable them with `additionalProperties: false` or
➥ constrain them[1].
➥ paths./community/api/v2/community/
➥ posts.post.requestBody.content.application/json.schema
```

Имя правила

JSONPath для нарушения правила

[1] Objects should not allow additionalProperties. Disable them with `additionalProperties: false` or constrain them. — Для объектов не должны быть разрешены дополнительные свойства (additionalProperties). Отключите их с помощью `additionalProperties: false` или ограничьте их. — *Примеч. пер.*

https://liveinternet.club

`owasp:api3:2023-no-default-additionalProperties` — кастомное правило линтинга API, которое я добавил в набор правил `.spectral.yaml`. Это правило создано в Team Digitale (https://mng.bz/67ro); оно проверяет, что для всех ответов свойство `additionalProperties` имеет значение `false`. Это правило и присваивание `additionalProperties:false` проверяет, что каждое поле, возвращаемое схемой ответа, документировано. Оно помогает перехватывать экземпляры, в которых злоумышленники пытаются спровоцировать API на аномальное поведение и вернуть непредвиденные данные.

> **ПРИМЕЧАНИЕ** Хотя присваивание `additionalProperties` значения `false` рекомендуется с точки зрения безопасности, оно затрудняет развитие API. Присваивание `additionalProperties:false` устанавливает ограничения на то, какие данные принимаются схемой, но оно также делает схему API жесткой и усложняет ее развитие, поскольку она не примет новые поля от сервера без обновления определения API. Кроме того, ключевое слово `additionalProperties` распознает только свойства, объявленные с ним в одной подсхеме, и это ограничивает возможности его использования с объединяющими ключевыми словами типа `allOf` и `anyOf`. Альтернативой для `additionalProperties` является ключевое слово `unevaluatedProperties`, которое решает эту проблему. Эта тема выходит за рамки книги, но вы сможете найти дополнительную информацию на сайте JSON Schema (https://mng.bz/oeDd). JSON Schema — спецификация схемы данных, используемая OpenAPI. Наконец, заметим, что в OpenAPI 2.0 `additionalProperties` по умолчанию присваивается значение `false`, а в OpenAPI 3.0 — значение `true`.

Но как насчет API, уже возвращающих поля, которые должны быть недоступны для пользователей? Это могут быть чувствительные поля, содержащие персональную информацию (PII, Personally Identifiable Information). Правило `owasp:api3:2023-no-default-additionalProperties` подобный сценарий не обнаружит. Такие поля можно пометить как чувствительные и настроить правило линтинга так, чтобы вы получали уведомления о том, где эти чувствительные поля возвращаются в ответе.

> **СОВЕТ** Здесь важно разобраться, какие поля считать чувствительными. Они должны быть задокументированы, а информация об этом доведена до всех команд, участвующих в разработке API.

Я создал правило `owasp:api3:2023-sensitive-fields-in-response`. Теперь проверьте отчет линтинга о вхождениях этого правила, выполнив следующую команду:

```
grep -E 'owasp:api3:2023-sensitive-fields-in-response' issues.log
```

Команда выдает следующий список ошибок:

```
2176:15
➥ warning
➥ owasp:api3:2023-sensitive-fields-in-response
➥ Possible sensitive fields in response. Field[1]: email
➥ components.schemas.Author.properties.email
```

Чувствительное поле в схеме ответа

[1] Possible sensitive fields in response. Field: — Ответ может содержать конфиденциальные поля. Поле: — *Примеч. пер.*

https://liveinternet.club

```
2178:19
➥ Warning
➥ owasp:api3:2023-sensitive-fields-in-response
➥ Possible sensitive fields in response. Field: vehicleid
➥ components.schemas.Author.properties.vehicleid
```

Чувствительное поле в схеме ответа

Объект схемы `components.schemas.Author` включается в объект `components.schemas.Post` и возвращается эндпоинтом `GET /community/api/v2/community/posts/recent`. Это означает, что любой пользователь, отправляющий запрос на получение последних постов с форума, также получает адрес электронной почты и `vehicleId` всех авторов, которые публиковали свои посты в последнее время. Эти поля не должны быть доступны всем пользователям; это проявление уязвимости нарушения авторизации на уровне свойств объектов. В табл. 10.2 показано, как выявлять эту уязвимость с помощью линтинга API.

Таблица 10.2. Обнаружение нарушений авторизации на уровне свойств объектов с помощью линтинга API

Уязвимость	Возможный эксплойт	Обнаружение с помощью линтинга API
Схема API допускает additionalProperties	Злоумышленник может попытаться спровоцировать API на неожиданное поведение, чтобы он ответил данными, не соответствующими определенной схеме, и раскрыл чувствительную информацию. Если API допускает недокументированные дополнительные свойства, он может вернуть поля, которые не должны возвращаться	Определите в файле OpenAPI все свойства объектов схемы. Задайте правило линтинга, которое проверяет, что additionalProperties присвоено значение false, где это возможно. Задайте правило линтинга, которое оповещает о наличии конфиденциальных полей в ответе, например о наличии персональной информации (электронной почты, номера телефона, номера кредитной карты и т. д.)
В некоторых операциях API отсутствуют ошибки 404 (не найдено), 415 (неподдерживаемый тип данных) и 500 (внутренняя ошибка сервера)	Злоумышленник может попытаться спровоцировать API на неожиданное поведение, чтобы получить больше информации о системе. Если стандартные ошибки не определены, может быть неясно, как API ведет себя в таких сценариях	Задайте правило линтинга, которое проверяет, что код состояния 404 (не найдено) определен для всех операций GET, PUT, HEAD и DELETE. Задайте правило линтинга, которое проверяет, что код состояния 415 (неподдерживаемый тип данных) определен для операций POST, PUT и PATCH. Задайте правило линтинга, которое проверяет, что код состояния 500 (внутренняя ошибка сервера) определен для всех операций API. Задайте правило линтинга, которое проверяет, что если для операции определен объект требований безопасности OpenAPI, то для нее также определены коды состояния 401 (не авторизован) и 403 (запрещено)

https://liveinternet.club

10.2.5. API4:2023. Неограниченное потребление ресурсов

Обслуживание запроса API требует затрат ресурсов на стороне провайдера API: памяти, процессора, пропускной способности сети, дискового пространства и т. д. В зависимости от операции API оно также может потреблять такие ресурсы, как отправка СМС или электронной почты либо активация телефонного звонка. Все это сопряжено с затратами для провайдера API.

Уязвимость неограниченного потребления ресурсов возникает, когда у API нет ограничений относительно того, какие ресурсы потребляются, либо ограничения заданы некорректно. При отсутствии ограничений злоумышленник может инициировать запросы, потребляющие большое количество ресурсов, что в итоге приведет к отказу в обслуживании из-за истощения ресурсов или взрывного роста операционных затрат для провайдера. Примеры ограничений ресурсов — максимальное количество файловых дескрипторов и выделяемой памяти для веб-приложения, максимальный размер отправляемых файлов, количество записей, возвращаемых на страницу, ограничения затрат для сервисов сторонних провайдеров.

Ограничение частоты запросов, часто называемое троттлингом (throttling), устанавливает лимит на количество выдачи запросов клиентом за определенный промежуток времени. Ограничение частоты вводится для того, чтобы при повышении частоты запросов одним клиентом сервис мог реагировать на запросы других клиентов. При превышении частоты запросов должен возвращаться код состояния 429 (слишком много запросов). Код состояния 429, определенный в RFC 6585, должен иметь тело ответа с подробностями, объясняющими, что клиент отправил слишком много запросов, и может включать заголовок Retry-After, указывающий, как долго клиент должен ожидать, чтобы вновь их отправить.

> **ПРИМЕЧАНИЕ** Риск неограниченного потребления ресурсов — это изменение классификации риска API4:2019 «Нехватка ресурсов и ограничение частоты» в OWASP 2019.

Чтобы устранить этот риск, необходимо установить меры управления потреблением ресурсов — такие, как политика ограничения частоты для эндпоинтов, определение лимитов процессора и памяти для сервисов, ограничение размера полезной нагрузки и ограничение объема данных, которые могут быть получены запросом с учетом размера страницы или количества страниц.

Давайте проверим отчет о линтинге файла `crapi.api.yaml` для этой проблемы, выполнив следующую команду:

```
grep -E 'owasp:api4' issues.log
```

Вы получите список ошибок:

```
2363:26
➥ error
➥ owasp:api4:2019-string-limit
```

https://liveinternet.club

```
➥ Schema of type string must specify
➥ maxLength, enum, or const[1].
➥ components.schemas.ServiceRequests.
➥ properties.service_requests.items.properties.created_on
```

Правило `owasp:api4:2019-string-limit` требует установления ограничений для строковых полей, чтобы избежать истощения ресурсов злоумышленниками, пытающимися внедрить большие строковые данные в запросы API. Аналогичное правило существует для числовых полей. В табл. 10.3 показано, как выявлять эту уязвимость с помощью линтинга API.

Таблица 10.3. Обнаружение уязвимости неограниченного потребления ресурсов с помощью линтинга API

Уязвимость	Возможный эксплойт	Обнаружение с помощью линтинга API
В схеме типа `string` (строка) не ограничены значения, передаваемые в API, с помощью шаблона для строкового значения или ограничения длины	Злоумышленник может отправлять большие данные в запросах API, чтобы спровоцировать API на непредвиденное поведение	Задайте правила линтинга для проверки того, что в строковых схемах имеется четко определенное регулярное выражение или перечисление. Примеры проверочных регулярных выражений можно найти на странице https://mng.bz/ngDa. В строковых схемах также должно быть задано значение `maxLength`
В числовой схеме не заданы минимальное или максимальное значения		Задайте правила линтинга для проверки того, что для числовых схем определены минимальные и максимальные значения. Для числовых значений должен быть установлен формат (например, int32 или int64)
В схеме массива не задано максимальное количество элементов		Задайте правила линтинга для проверки того, что для массивов всегда определяется значение `maxItems`
API включает операции, для которых отсутствует ответ 329	Злоумышленник может запустить атаку на API методом брутфорса	Задайте правила линтинга для проверки того, что для всех операций, кроме HEAD, определен ошибочный ответ 429 (слишком много запросов). Ответы 429 должны определять заголовок Retry-After. Если операции HEAD должны выполняться в фоновом режиме, но не возвращают тело в ответе, также необходимо учитывать эффект потребления ресурсов для операции HEAD

[1] Schema of type string must specify maxLength, enum, or const — Для схемы типа string необходимо установить значения maxLength, enum или const. — *Примеч. пер.*

https://liveinternet.club

10.2.6. API5:2023. Нарушения авторизации на уровне функций

Эта уязвимость позволяет злоумышленнику выполнять несанкционированные операции. Пример уязвимости нарушений авторизации на уровне функций (BFLA, Broken Function Level Authorization) — возможность обращения к эндпоинтам администрирования и выполнения административных действий (например, создания пользователя) со стороны обычных пользователей. Другой пример — злоумышленник с пользовательской ролью, имеющей разрешение только на чтение ресурса (выполнение операции GET), получает возможность обновлять или удалять ресурсы, поскольку модуль авторизации не проверяет команду в запросе HTTP. В отличие от уязвимости BOLA, суть которой заключается в несанкционированном доступе к объектам за счет манипуляции с идентификаторами объектов, суть BFLA заключается в управлении доступом на уровне функций.

Так как эта уязвимость связана с авторизацией пользователей для конкретных операций, ее не удастся легко выявить с помощью линтинга API. Устранение этого риска требует проверки правильности проектирования и тестирования механизмов автризации в коде, чтобы операции могли выполнять только пользователи из надлежащей авторизованной группы или с определенной ролью.

10.2.7. API6:2023. Неограниченный доступ к чувствительным бизнес-потокам

Чувствительные бизнес-потоки с высоким влиянием на бизнес нередко становятся доступными через API. Например, процесс бронирования авиабилетов является чувствительным бизнес-потоком, который напрямую влияет на прибыль. Доступ к этому потоку может предоставляться через API бронирования билетов. При отсутствии ограничений на использование этого чувствительного бизнес-потока может возникнуть уязвимость. Например, если API бронирования билетов позволяет пользователям приобретать билеты без сбора за отмену, злоумышленник может воспользоваться этим обстоятельством и скупить бóльшую часть билетов на рейс при их появлении. Ближе к дате вылета злоумышленник отменяет все покупки, заставляя авиакомпанию распродавать билеты со скидкой. Злоумышленник пользуется этим и покупает один билет по низкой цене. Как видите, эксплуатация уязвимостей в чувствительных бизнес-потоках требует хорошего понимания бизнес-модели, процессов и слабостей в ограничениях на использование.

Эту уязвимость невозможно выявить с помощью линтинга, хотя проверка наличия ограничений на частоту у эндпоинтов может оказаться эффективной. Тем не менее это пример того, как качественное моделирование угроз помогает заранее увидеть наличие риска. Чтобы его устранить, необходимо определить, какие бизнес-потоки могут быть ему подвержены и какие средства контроля можно добавить для их защиты. Необходимо организовать мониторинг использования

https://liveinternet.club

API и ограничивать подозрительное поведение клиентов, использующих определенные устройства или IP-адреса. Для этого могут пригодиться капча и другие средства проверки человеческого участия.

10.2.8. API7:2023. Фальсификация запросов на стороне сервера

Фальсификация запросов на стороне сервера, или SSRF (Server-Side Request Forgery), — уязвимость, позволяющая злоумышленнику выполнить запрос по непредусмотренному внутреннему или внешнему адресу. Это может быть использовано для доступа к иначе недоступной внутренней системе с целью извлечения секретной информации (например, учетных данных для авторизации).

Такая уязвимость встречается в сценариях, когда пользователю разрешается предоставить URL-адрес, к которому отправляет запрос сервер. Если сервер не проведет должную проверку URL, прежде чем обращаться к нему с запросом, злоумышленник может сконструировать URL, приводящий к непредвиденным последствиям. Например, приложения социальных сетей могут разрешать пользователям либо напрямую загружать фотографии профиля, либо предоставлять URL-адреса, с которых эти фотографии могут быть получены. Злоумышленник может использовать это, указав адрес адрес вида `http://localhost:8080`, чтобы выполнить сканирование портов, и если открытый порт будет обнаружен — провести дальнейшие атаки.

Меры по снижению риска включают запрет на перенаправление HTTP-серверу API там, где это возможно, создание белого списка разрешенных адресов и перенаправлений, а также ограничение диапазона схем URL и разрешенных портов.

10.2.9. API8:2023. Неверная конфигурация безопасности

Эта уязвимость возникает, когда неверная конфигурация сервера API раскрывает конфиденциальные данные и позволяет злоумышленнику получить несанкционированный доступ к серверу или нарушить его работу. Появление таких уязвимостей может быть вызвано любой из следующих причин на сервере API: отсутствием последних патчей безопасности, небезопасными конфигурациями по умолчанию, неправильной настройкой разрешений в облачных сервисах, неверной конфигурацией заголовков HTTP, разрешением необязательных команд HTTP, отсутствием TLS, неправильной настройкой политик CORS или разрешением включать в сообщения ошибок трассировку стека и другую конфиденциальную информацию. Неверная конфигурация сервера API может возникнуть в ходе реализации, выделения сред или развертывания.

> **ПРИМЕЧАНИЕ** В API Security Top 10 за 2019 год этот риск назывался API7:2019 «Неверная конфигурация безопасности».

Меры по снижению риска включают автоматизацию процессов усиления защиты и установки патчей для кода, библиотек и контейнеров; отключение лишней

https://liveinternet.club

функциональности; автоматизацию тестирования эндпоинтов API на правильность конфигурации. Одна из областей уязвимости неправильной конфигурации безопасности, которую может выявить линтинг API, — использование HTTP вместо HTTPS. В качестве примера выполните следующую команду:

```
grep -E 'owasp:api7' issues.log
```

Команда находит одну проблему с помощью линтинга:

```
9:10
➥ error
➥ owasp:api7:2019-security-hosts-https-oas3
➥ Server URLs MUST begin https://, and no other protocol is permitted[1].
➥ servers[0].url
```

Как видите, правило `owasp:api7:2019-security-hosts-https-oas3` проверяет, что серверы используют HTTP, а не HTTPS. В табл. 10.4 показано, как выявлять эту уязвимость с помощью линтинга API.

Таблица 10.4. Обнаружение уязвимости неверной конфигурации безопасности с помощью линтинга API

Уязвимость	Возможный эксплойт	Обнаружение с помощью линтинга API
API получает незашифрованные коммуникации HTTP	Злоумышленник может прослушивать сетевой трафик с перехватом всех запросов и ответов	Задайте правило линтинга для проверки того, что URL сервера используют HTTPS, а не HTTP

10.2.10. API9:2023. Некорректное управление учетом ресурсов

Эта уязвимость возникает, когда из-за недостаточной информации о ресурсах API организация предоставляет неисправленные или незащищенные API, которые могут стать целью злоумышленников. Недостаток информации о ресурсах API или устаревшая документация усложняют поиск и исправление уязвимостей. Например, может не хватать информации о том, в какой среде работает API, какая версия API актуальна, какой уровень пользовательского доступа должен предоставлять API или каковы планы вывода из эксплуатации старых версий. Механизмы безопасности и программные исправления новейших версий API могут отсутствовать в старых версиях. Злоумышленник может обнаружить старую, неисправленную версию API и получить доступ к конфиденциальным данным или взять под контроль сервер. Эта уязвимость связана с практикой управления API в организации.

[1] Server URLs MUST begin https://, and no other protocol is permitted — URL-адреса серверов ДОЛЖНЫ начинаться с https://, использование других протоколов не допускается. — *Примеч. пер.*

https://liveinternet.club

Поддержание актуальной информации об API в сочетании с выводом из эксплуатации старых версий API поможет снизить этот риск. Также вам помогут брандмауэры, жесткие механизмы аутентификации и сегрегация доступа к эксплуатационным и неэксплуатационным данным.

10.2.11. API10:2023. Небезопасное потребление API

API могут взаимодействовать с другими сторонними API и зависеть от них. Разработчики обычно внимательно обращаются с вводом пользователя, проверяют и очищают эти данные. Однако они могут доверять данным, полученным от сторонних API известных компаний и проявлять меньшую строгость при их проверке. Когда API безоговорочно доверяют ответам от других API, возникает уязвимость небезопасного потребления API. Она проявляется, когда API взаимодействуют с другими сторонними API по незашифрованным каналам или слепо следуют сторонним перенаправлениям. Если сторонний API оказался под контролем злоумышленника, он может использоваться для атаки других зависимых API.

Например, если сторонний адресный API был взломан и злоумышленник сохранил данные SQL-инъекции (SQLi) под видом своего бизнес-адреса, он может отправить запрос к уязвимому API, который зависит от адресного API, и заставить его загрузить вредоносный адрес. Уязвимый API может выполнить код в своей базе данных, что приведет к потере данных или раскрытию конфиденциальной информации.

Меры снижения этого риска включают использование защищенных каналов связи для сторонних коммуникаций. Ввод от сторонних API должен тщательно проверяться во время выполнения — например, в коде приложения или API-шлюзе. Эти данные, так же как и ввод от пользователя, должны проходить фильтрацию и очистку для предотвращения атак инъекции.

На этом наше обсуждение «верхней десятки» уявимостей API от OWASP подходит к концу. Как видите, чтобы использовать линтинг API для выявления уязвимостей безопасности в определениях OpenAPI, важно иметь определение OpenAPI с точно и подробно документированной схемой полезной нагрузки запросов и ответов. Это позволит выполнять правила линтинга безопасности API (наряду с другими правилами линтинга) в редакторе проектирования API и, что еще важнее — в пайплайне CI-CD.

> **СОВЕТ** Мы рассмотрели проверки линтинга (такие, как Spectral OWASP) для поиска уязвимостей в определениях API. Если вы пользуетесь VS Code как основным текстовым редактором, не забывайте, что для сканирования файлов определений API на предмет уязвимостей также можно применять плагин 42Crunch OpenAPI (Swagger) Editor (установка которого рассматривается в приложении Б).

https://liveinternet.club

Мы рассмотрели пять уязвимостей, отдельные признаки которых можно выявлять с помощью линтинга определений OpenAPI: нарушения авторизации на уровне объектов, нарушения аутентификации, нарушения авторизации на уровне свойств объектов, неограниченное потребление ресурсов и неправильную конфигурацию безопасности. Также мы рассмотрели пять других уязвимостей, для обнаружения и устранения которых потребуются другие средства: нарушения авторизации на уровне функций, неограниченный доступ к чувствительным бизнес-потокам, фальсификация запросов на стороне сервера (SSRF), некорректное управление учетом ресурсов и небезопасное потребление API. В следующем разделе поговорим о том, как использовать файлы определения OpenAPI для обеспечения защиты на уровне API-шлюза.

10.2.12. Проверка трафика в шлюзе с использованием OpenAPI

Если у вас есть точный документ определения OpenAPI, его можно передать на API-шлюз и использовать для проверки входящих запросов до того, как запрос достигнет микросервисов бэкенда. Некоторые API-шлюзы (например, Amazon API Gateway, Kong с плагином OAS Validation (https://mng.bz/v8D7) и Apigee с политикой OASValidation (https://mng.bz/4JEw)) позволяют это сделать. Запросы, не соответствующие заданной схеме, получат от шлюза ответ HTTP 400 Bad Request (неправильный запрос). Таким образом снижается нагрузка на бэкенд-сервисы, а у вас появляется централизованное место для проверки запросов.

Кроме того, шлюз может поддерживать проверку ответов, гарантируя, что микросервис не вернет никаких непредвиденных свойств данных, схема которых не указана в определении OpenAPI. Для плагина Kong OAS Validation, если проверка ответа завершается неудачей, шлюз возвращает код состояния HTTP 406 Not Acceptable (неприемлемо). Помимо наружного трафика, проверка запросов и ответов через API-шлюз может использоваться для защиты системы при взаимодействии со сторонними сервисами — например, от уязвимости небезопасного потребления API, рассмотренной ранее. В этом сценарии можно разместить API-шлюз (или прокси-сервер, или промежуточное ПО с поддержкой проверки OpenAPI) между вашей платформой и третьей стороной, и проверять трафик на соответствие определению OpenAPI, предоставляемому этой третьей стороной (рис. 10.6). Также можно настроить шлюз на отправку оповещений системе мониторинга в случае неуспешной проверки.

СОВЕТ Помимо проверки контракта OpenAPI, шлюз или прокси-сервер для сторонних API, потребляемых сервером, может выполнять другие функции — например, предоставление централизованного места для визуализации и мониторинга объема, производительности и частоты ошибок сторонних API, включая управление квотами запросов API для третьих сторон. Пример специализированного прокси-сервера, который решает эти задачи, — Lunar.dev (https://github.com/TheLunarCompany/lunar).

https://liveinternet.club

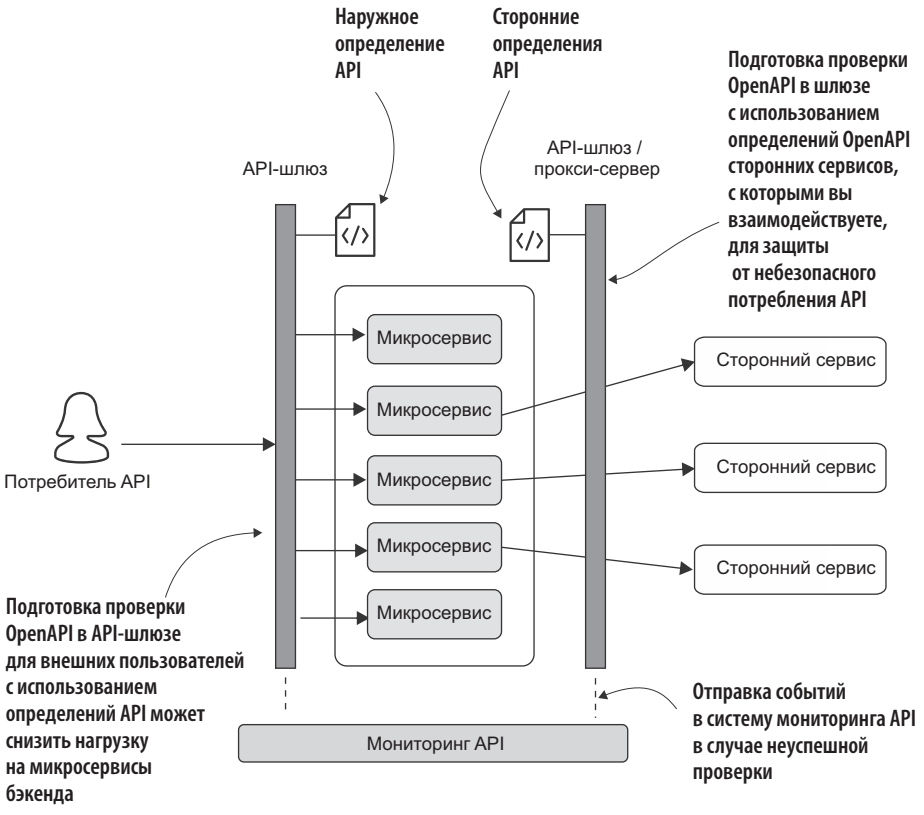


Рис. 10.6. Проверка запросов в шлюзе с использованием OpenAPI

Конечно, чтобы схема проверки на шлюзе работала, он должен иметь определение OpenAPI, которое мог бы использовать. У организаций оно есть не всегда, что приводит нас к следующей теме.

10.2.13. Генерация определений API на основе трафика

Мы обсудили проверки безопасности API на основе файла определения API для выявления уязвимостей с помощью линтинга. Но что, если для вашего API определение OpenAPI недоступно? Многие организации используют API, не имеющие качественной документации. В таком случае начать стоит с документирования API — трудно защитить то, о чем вы ничего не знаете. Для этого можно сгенерировать определения API на основе кода приложения там, где это возможно (как обсуждалось в главе 6). Также можно перехватить трафик API и воспользоваться им для генерации определения API. Существует несколько вариантов реализации второго способа.

https://liveinternet.club

Можно воспользоваться инструментом, который сделает все сразу, — например, Optic (https://github.com/opticdev/optic). Optic может работать как посредник, который перехватывает трафик API (например, генерируемый тестами API или коллекциями Postman), а затем создает из него определение OpenAPI. Подробнее см. на странице www.useoptic.com/docs/generate-openapi-from-tests.

Кроме того, можно воспользоваться специализированным прокси-сервером — таким, как Charles HTTP Proxy (www.charlesproxy.com/), — для перехвата трафика (также генерируемого тестами API или другими источниками трафика), а затем экспортировать сеанс трафика в файл формата HAR (HTTP ARchive). Har2OpenAPI (https://github.com/dcarr178/har2openapi) — инструмент с открытым исходным кодом, который может преобразовать HAR-файл в определение OpenAPI.

Третий вариант основан на использовании платформы безопасности API с соответствующей встроенной функциональностью. Платформы безопасности API рассматриваются в следующем разделе.

10.2.14. Платформы безопасности API

Безопасность API — сложная и стремительно изменяющаяся область. Существуют специализированные платформы безопасности API (такие, как 42Crunch, NoName, Salt Security и Traceable AI), которые помогают организациям защитить свои API. Они предназначены для интеграции безопасности API в общий жизненный цикл разработки API. Многие из них также обладают возможностями машинного обучения, позволяющими узнавать об уязвимостях и эксплойтах в сети клиентов, чтобы автоматически обнаруживать и применять защиту на основании полученной информации. Вы как провайдер API должны проанализировать преимущества интеграции с разработчиками платформ безопасности API.

В широком смысле платформы безопасности API обеспечивают защиту на трех основных этапах жизненного цикла разработки API: (1) на этапе проектирования предоставляют средства сканирования определений API на наличие уязвимостей; (2) на этапе тестирования — способы динамического тестирования API приложений на наличие уязвимостей и атак бизнес-логики; (3) на этапе запуска в продакшен — механизм мониторинга живого трафика на наличие угроз, позволяя провайдерам API идентифицировать атаки и предпринимать действия по устранению угроз, например блокировку IP-адреса или учетной записи злоумышленника. Мониторинг трафика API осуществляется добавлением агентов инструментации к API-шлюзам и микросервисным приложениям, что делает возможной распределенную трассировку и анализ данных в потоках запросов и ответов (обычно с анонимизацией чувствительной информации). Также платформы способны обнаруживать *теневые API* (недокументированные API, которые существуют и работают вне официально наблюдаемых каналов организации; иногда их создают разработчики, чтобы сэкономить время)

https://liveinternet.club

и *зомби-API* (устаревшие, забытые API, которые все еще работают в фоновом режиме и теоретически могут быть использованы для атак).

Мы рассмотрели проверку определений API на наличие уязвимостей безопасности, а также возможности, предоставляемые платформами безопасности API. Перейдем к общему обзору способов защиты приложений за счет интеграции безопасности в процесс создания продукта.

10.3. DevSecOps

Как уже говорилось, моделирование угроз — хорошая отправная точка для проектирования безопасных приложений. Но при этом команды должны учитывать полный жизненный цикл разработки приложения. И здесь пригодится DevSecOps (Development, Security, Operations) — методология, расширяющая DevOps для интеграции инициатив безопасности на протяжении жизненного цикла приложения с целью создания безопасных приложений. Она развивает культуру общей ответственности за безопасность для всех, кто участвует в создании и эксплуатации программного обеспечения.

Кроме моделирования угроз, DevSecOps включает следующие практики.

- *Статическое тестирование безопасности приложений* (SAST, Static Application Security Testing) — анализ исходного кода в среде, не связанной с исполнением, на ранней стадии процесса разработки для проверки уязвимостей. Примеры инструментов SAST — Checkmarx и Snyk.
- *Анализ программных компонентов* (SCA, Software Component Analysis) — автоматизация видимости в продуктах с открытым исходным кодом, используемых при создании приложений, для обеспечения управления рисками, безопасности и соответствия условиям лицензирования.
- *Интерактивное тестирование безопасности приложений* (IAST, Interactive Application Security Testing) — взаимодействие с работающими приложениями и их анализ (с помощью автоматизированного теста или человеком-тестировщиком) для выявления потенциальных уязвимостей.
- *Динамическое тестирование безопасности приложений* (DAST, Dynamic Application Security Testing) — тестирование по принципу «черного ящика» с моделированием атак и проверкой безопасности приложений из-за границ сети. Инструменты DAST автоматически обходят и тестируют API на предмет SQLi, межсайтовых скриптовых атак (XSS), межсайтовой фальсификации запросов (CSRF) и других уязвимостей. Примеры инструментов DAST — InsightAppSec от Rapid7 и Burp Suite от PortSwigger.
- *Фаззинг-тестирование* — внедрение недействительных, некорректно сформированных или непредвиденных входных данных в программную систему для выявления уязвимостей и возможных последствий.

https://liveinternet.club

- *Сканирование уязвимостей контейнеров* — сканирование контейнеров с целью обнаружить присутствие вредоносного ПО, устаревших библиотек и неверных конфигураций.
- *Подписи образов* — добавление цифровых подписей в образ контейнера для обеспечения его подлинности и целостности.
- *Управление секретами* — использование инструментов и методов управления цифровыми данными: паролями, токенами доступа и т. д. в приложениях и привилегированных учетных записях. Примеры инструментов управления секретами — Vault (HashiCorp) и AWS Secrets Manager.
- *Сканирование инфраструктурных уязвимостей* — сканирование инфраструктуры (аппаратных и программных компонентов: серверов, коммутаторов, брандмауэров, маршрутизаторов, накопителей, операционных систем и т. д.) и выполняемых приложений на наличие уязвимостей.
- *Самозащита приложений во время выполнения* (RASP, Runtime Application Self-Protection) — технология безопасности, внедряемая в приложения, которые используют инструментацию времени выполнения и контекстную информацию для мониторинга, обнаружения и автоматической блокировки атак во время выполнения.

На рис. 10.7 показано, какое место эти практики занимают в пайплайне DevSecOps.

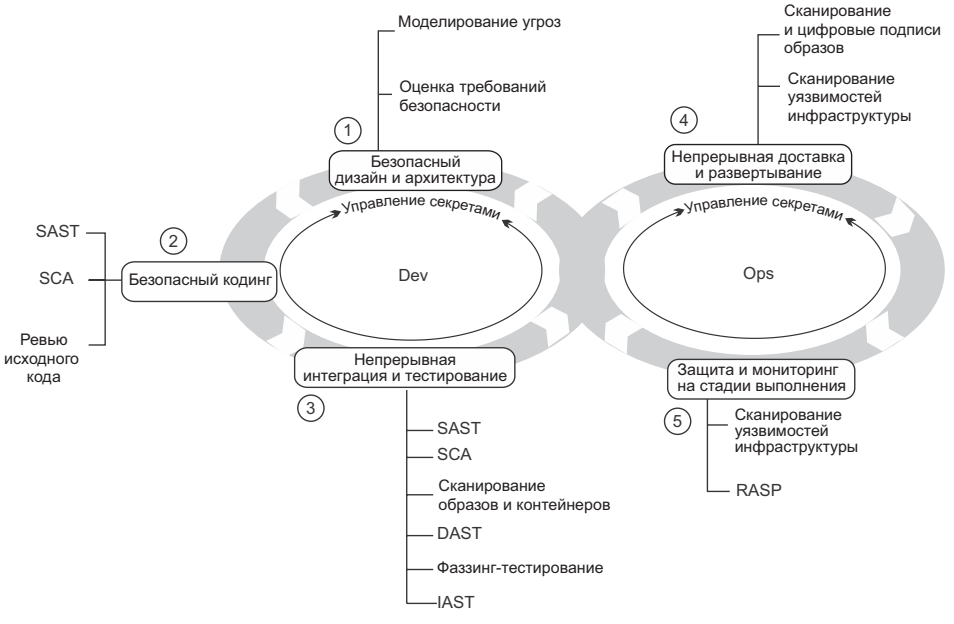


Рис. 10.7. Пример пайплайна DevSecOps

https://liveinternet.club

Как видите, защита API подразумевает не только их тщательное проектирование с учетом безопасности, но и внедрение элементов безопасности через пайплайн развертывания.

На этом мы завершаем обсуждение вопросов безопасности. Вкратце напомним основные выводы этой главы.

Итоги

- Для соблюдения некоторых рекомендаций по стилю API может потребоваться определить собственные (кастомные) функции. Собственная функция получает целевой объект, для которого необходимо выполнить проверки, и позволяет пользоваться для этих проверок всеми возможностями языка программирования, на котором работает выбранный инструмент проверки кода.
- Для собственных функций следует писать юнит-тесты. Но прежде чем принимать решение об использовании таких функций, проверьте, нет ли в вашем инструменте линтинга API встроенной функции или правила, которые делают то, что вам нужно. Также проверьте, существуют ли общедоступные правила линтинга, которые можно импортировать для решения нужной задачи.
- При проектировании API необходимо предусмотреть защиту от угроз безопасности, с которыми может столкнуться ваш API. Список OWASP API Security Top 10 составлен отраслевыми экспертами и содержит общий обзор самых распространенных угроз безопасности API. Встроенные в правила линтинга API проверки безопасности помогут обнаружить уязвимости на этапе проектирования.
- Такие методы, как сравнение трафика API во время выполнения с определением OpenAPI в API-шлюзе, помогут снизить нагрузку на сервисы бэкенда, что формирует уровень валидации и помогает повысить безопасность API. Этот метод также можно использовать для проверки трафика третьих сторон (на основе их определений API) для защиты от уязвимостей, относящихся к небезопасному потреблению сторонних API.
- Помимо проектирования API для безопасности, рассмотрите возможность интеграции с платформами безопасности API для реализации средств контроля в жизненном цикле разработки API. Также для защиты приложений следует интегрировать практики DevSecOps в пайплайн развертывания.

https://liveinternet.club

11 Мониторинг и аналитика: метрики продукта API

В этой главе

- ✓ Аналитика продукта API: использование API, время до первого вызова, активные потребители
- ✓ Метрики мониторинга API: задержка, трафик, частота ошибок

В главе 1 вы узнали, что термином APIOps обозначается сквозная автоматизация жизненного цикла разработки, основанная на принципах DevOps и GitOps. Важным принципом DevOps, в представлении Джина Кима, является принцип непрерывной обратной связи для обнаружения и исправления проблем в потоке создания ценности ПО. Один из способов его реализации — практика DevOps по непрерывному мониторингу и измерениям. Она направлена на сбор, анализ и обмен метриками из продакшен-системы с командой разработки для формирования цикла улучшений. Это подразумевает использование заранее определенных метрик, дашбордов и оповещений для понимания и анализа поведения работающих систем, а также решение проблем.

Помимо технической производительности API, команда разработки и владельцы продукта API нуждаются в непрерывной обратной связи по *метрикам продукта* API. Метрики продукта API измеряют опыт взаимодействия с продуктом API в контексте общей продуктовой стратегии. Они помогают оптимизировать эффективность продукта и согласовать действия клиентов с бизнес-целями. С помощью этих метрик менеджеры продуктов могут оценивать успех отдельных

https://liveinternet.club

функций продукта. Метрики также помогают команде API принимать обоснованные решения относительно возможных улучшений продукта для достижения бизнес-целей.

В этой главе рассматривается применение мониторинга и аналитики продукта на эксплуатационном этапе API. На рис. 11.1 показано, какое место они занимают в жизненном цикле разработки API.



Рис. 11.1. Мониторинг API и аналитика продукта находятся на эксплуатационном этапе жизненного цикла доставки продукта API

В примерах этой главы для мониторинга и аналитики используется инструмент Moesif. В приложении Г приведены инструкции по созданию учетной записи Moesif.

11.1. Аналитика продукта API: бизнес-задача

Допустим, ваш API каталога продуктов развернут на продакшен. Он хорошо работает, но руководство установило амбициозную цель на следующий квартал:

https://liveinternet.club

увеличить на 10 % количество клиентов, пользующихся API. Для достижения этой цели вашей команде поставлена задача улучшить общий опыт взаимодействия с продуктом (относительно как самого API, так и его документации и маркетинга). Как это сделать?

Чтобы объективно подойти к решению задачи, вам понадобятся данные о текущем опыте взаимодействия с продуктом. Они позволят принять решение и выбрать эталонные значения для мер по улучшению продукта. В этой главе вы узнаете, как собрать соответствующие метрики. Но сначала я объясню, как данные аналитики продукта используются для управления улучшениями в цикле PDSA (Plan-Do-Study-Act).

11.1.1. Разработка продукта API как цикл PDSA

В главе 2 был представлен цикл PDSA — итеративный метод введения изменений и улучшений. Как отмечалось, цикл состоит из четырех стадий (планирование, выполнение, изучение и действие). Разработка продукта рассматривается не как линейный процесс с заданным началом и завершением, а как исполнение цикла PDSA по усовершенствованию продукта. Полученная информация, включая метрики эффективности продукта по каждой завершенной итерации, используется для запуска следующей итерации.

Как это работает? На стадии планирования руководство компании обычно выбирает одну высокоуровневую бизнес-метрику (из множества отслеживаемых), которая должна быть улучшена в течение заданного периода. Бизнес-метрики могут относиться к какому-то количественному показателю (например, увеличение среднего чека в интернет-магазине на 20 % или рост числа активных подписчиков на 10 %), качественному показателю (например, повышение удовлетворенности или удержания клиентов) либо к сокращению затрат.

Такая центральная метрика, или *фокус-метрика* (также называемая метрикой Полярной звезды), должна обладать ценностью для клиента и бизнеса. На ее основе менеджеры продукта выявляют и отслеживают низкоуровневые метрики, которые должны улучшать эту метрику высшего уровня либо поддерживать общую работоспособность продукта в процессе внедрения улучшения.

Как правило, для API такими низкоуровневыми метриками становятся метрики продукта API в областях *обнаружения* API, *вовлечения*, *активации*, *принятия*, *удержания* и *взаимодействия*. Позже я объясню смысл этих терминов, но их можно рассматривать как метрики, отслеживающие причинно-следственные действия, с помощью которых обеспечивается ожидаемый высокоуровневый результат. Предиктивная теория менеджеров продукта (причинно-следственная связь) гласит, что улучшение этих низкоуровневых метрик (например, ежедневного количества активных пользователей — метрики, которую мы обсудим далее) должно, в свою очередь, улучшить высокоуровневую целевую метрику, интересующую бизнес. На основании этой гипотезы команда API планирует

https://liveinternet.club

улучшения — новые API или новые возможности продукта API, — которые приводят к улучшению фокус-метрики продукта API.

Теперь вы знаете, что собой представляет стадия планирования цикла PDSA, и я кратко опишу другие стадии.

- Стадия выполнения (Do) включает реализацию, развертывание и публикацию улучшений продукта API.
- Стадия изучения (Study) — это измерение метрик продукта API со временем для проверки гипотезы. Также на этой стадии делаются выводы из успеха или неудачи эксперимента.
- Стадия действия (Act) представляет собой широкое распространение сделанных выводов. Если изменения были успешными, выявляются точки действий для повсеместного принятия и точки действий для улучшений на следующей итерации. Если изменения были неуспешными, следует выяснить, почему они не привели к ожидаемому результату, а затем спланировать действия для дальнейшего усовершенствования. После этого в обоих случаях цикл повторяется снова (рис. 11.2).

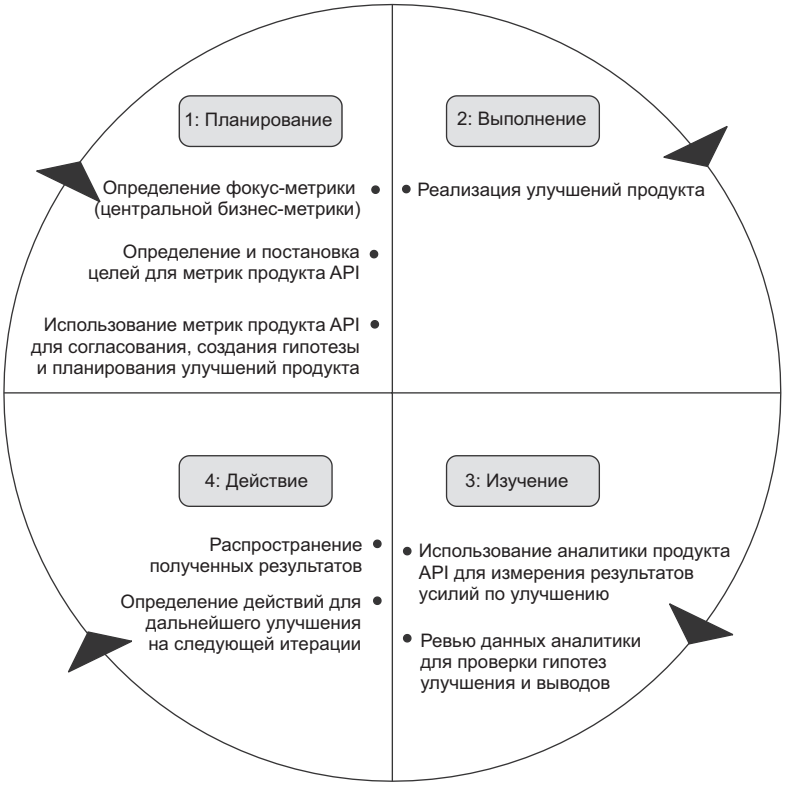


Рис. 11.2. Улучшения продукта API в цикле PDSA

https://liveinternet.club

Итеративный цикл PDSA подчеркивает важность аналитики продукта API. Возможность сохранения количественных данных для проверки или опровержения гипотезы улучшения продукта — ключ к созданию успешных продуктов API. Но прежде чем переходить к подробному обсуждению метрик, рассмотрим концепцию, которая дает представление о том, где именно должны отслеживаться метрики, — *карту пути разработчика* (developer journey map).

11.1.2. Карта пути разработчика

В главе 5 мы рассмотрели карту пути пользователя (user journey map) как наглядное представление процесса, через который проходит пользователь при взаимодействии с продуктом для выполнения своей задачи, или работы (в терминологии JTBD — «работа, которая должна быть выполнена»). Момент, когда пользователь выполнил работу, которую планировал, — это тот самый момент, когда он осознает, что продукт принес ему пользу. Иногда это называют моментом ценности (value moment). Сбор событий или действий в таких точках помогает измерить практическую ценность продукта.

Для менеджеров API-продуктов разработчики, интегрирующиеся с API, — это основные пользователи, и у карты пути пользователя появляется новая разновидность: *карта пути разработчика*. Она наглядно представляет путь, который проходит разработчик от обнаружения продукта API до интеграции с ним и взаимодействия с зависимыми от него рабочими системами. Обычно этот путь состоит из стадий *обнаружения*, *оценки*, *сборки*, *эксплуатации* и *масштабирования* продукта API; работа, которую планирует выполнить разработчик на каждой стадии, представлена на рис. 11.3.

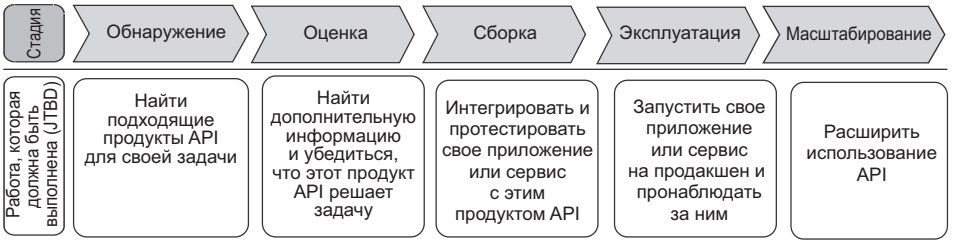


Рис. 11.3. Начало карты пути разработчика с заданиями, которые разработчик стремится выполнить на каждой стадии

> **ПРИМЕЧАНИЕ** Эти стадии используются в примерах книги, но в своей организации вы можете выделить те, которые лучше отражают ви́дение пути разработчика, принятое в ней.

Карта также включает *точки контакта разработчика* (touchpoints) для каждой стадии, то есть взаимодействия разработчика с вашим продуктом API. Это

https://liveinternet.club

точки, где вы можете внедрить сбор метрик продукта API для измерения опыта взаимодействия и эффективности продукта с точки зрения клиента.

Карта пути разработчика помогает менеджерам продуктов API и специалистам по связям с разработчиками (developer relations) получить представление об опыте взаимодействия с API с точки зрения разработчика. На рис. 11.4 показан пример карты пути разработчика с указанием точек контакта.

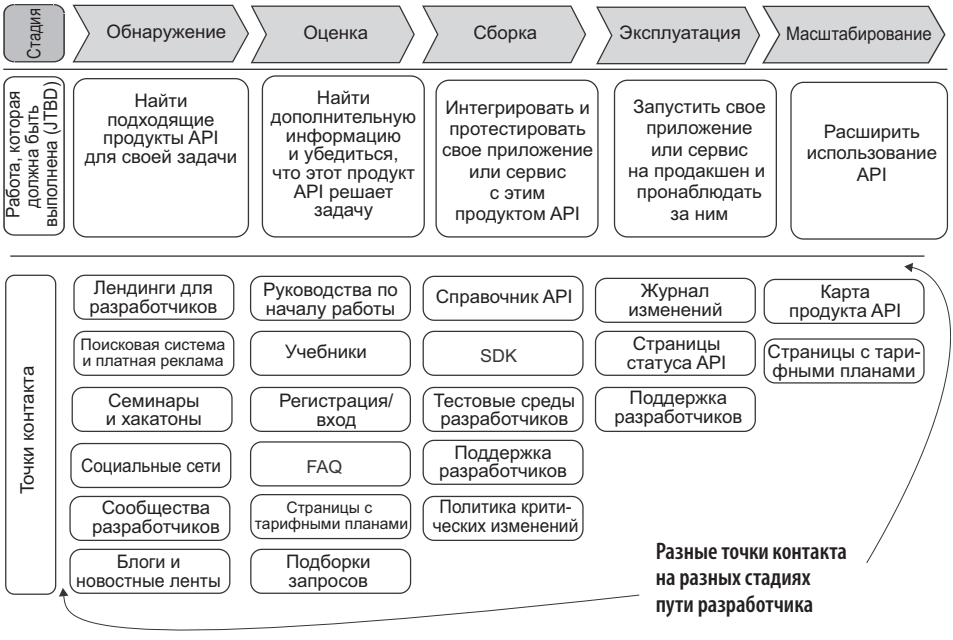


Рис. 11.4. Карта пути разработчика, дополненная точками контакта

Менеджеры продуктов API стараются, чтобы разработчики добрались от левого края карты до правого как можно быстрее и с минимальными препятствиями. Для привлечения внимания или ускорения принятия продукта API некоторые менеджеры дают возможность разработчикам опробовать API, чтобы они оценили его пользу и предложили его своему руководству. Такой подход к расширению принятия продукта API называется принятием, ориентированным на разработчика (developer-first adoption), или подходом, ориентированным на разработчика (developer-first approach), то есть «сначала разработчик».

Как уже говорилось, когда разработчик выполняет намеченную работу на каждой стадии своего пути, появляется точка (момент времени), в которой ваш продукт приносит ему практическую ценность. В этой точке можно сохранить метрики API-продукта, чтобы измерить доставленную ценность, а также проследить за тем, как эти метрики влияют на фокус-метрику, определенную руководством. В следующем разделе мы обсудим метрики продуктов API и их классификацию.

https://liveinternet.club

Подход, ориентированный на разработчика, и подход с вовлечением разработчика

Некоторые организации создают продукты, предназначенные прежде всего для разработчиков, и продвигают их методом, ориентированным на разработчика, — они предлагают разработчикам принять продукт и использовать его с низкими затратами и минимальными препятствиями, а затем переходят к основной модели продаж. Другие организации создают продукты, в первую очередь направленные на бизнес или потребителей. Но и в таких продуктах могут быть компоненты (например, API), доступные для разработчиков. В этом случае организации используют модель с вовлечением разработчика (developer-plus) и пытаются привлечь разработчиков как партнеров — например, стимулируя их создавать приложения для предлагаемого API или привлекая их через маркетплейс. Некоторые организации применяют гибридные решения. Различия в подходах означают, что менеджеры продуктов API могут применять разные стратегии для стимулирования принятия. Подробнее эта тема рассмотрена в книге «Developer Relations: How to Build and Grow a Successful Developer» Program Кэролайн Левко (Caroline Lewko) и Джеймса Партона (James Parton) (Apress, 2021).

11.1.3. Метрики продукта API

В своей книге «API Analytics for Product Managers» (Pakt Publishing, 2021) Дипа Гойял (Deepa Goyal) определяет шесть категорий метрик API-продуктов, отслеживаемых менеджерами продукта: *метрики обнаружения*, *метрики вовлеченности*, *метрики привлечения*, *метрики активации*, *метрики удержания* и *метрики опыта взаимодействия*. Как отмечалось в разделе 11.1.1, эти метрики являются метриками действий, которые влияют (и, следовательно, являются ведущими индикаторами) на фокус-метрику, определенную руководством. В следующих разделах мы обсудим некоторые метрики из этих категорий, но следует помнить, что это далеко не все возможные метрики.

ПРИМЕЧАНИЕ Далее под термином *«пользователи»* понимаются разработчики, а под термином *«конечные пользователи»* — потребители, которые используют приложения, созданные разработчиками и интегрируемые с API. Как уже говорилось, менеджеры продуктов API рассматривают разработчиков как своих основных пользователей. Даже при том, что разработчик может создавать приложение, к которому обращаются конечные пользователи, для целей метрик продуктов API мы классифицируем все вызовы API, совершаемые из приложения (и связанного с ним ключа API), как исходящие от этого разработчика независимо от количества конечных пользователей приложения.

Метрики обнаружения

Метрики обнаружения (discovery metrics) отслеживают эффективность маркетинговых каналов, используемых менеджерами продуктов API для распространения осведомленности о продукте среди пользователей. Примеры таких каналов — поисковые системы, лендинги для разработчиков, социальные сети,

https://liveinternet.club

маркетплейсы API и т. д. Метрики обнаружения обычно определяются инструментами веб-аналитики (например, Google Analytics и Mixpanel) на лендингах портала разработчика.

Примеры таких метрик:

- количество *уникальных посетителей* лендинга за день, неделю или месяц;
- количество *просмотров* лендинга;
- количество *рефералов из различных источников трафика* (или регистраций учетных записей разработчиков из реферального канала) как метрики того, какие маркетинговые каналы наиболее эффективны для обнаружения API.

> **СОВЕТ** Инструменты веб-аналитики (например, Google Analytics или Amplitude) на портале разработчика позволяют получать непрерывную обратную связь о взаимодействиях пользователей. Некоторые порталы разработки (например, Readme) содержат встроенные функции аналитики (https://mng.bz/d6GD). Если вы публикуете коллекции в открытых рабочих пространствах Postman или в сети Postman API, в них также существуют встроенные средства аналитики, которые помогают собирать метрики принятия вашего API. Вы узнаете, сколько раз пользователи просматривали вашу коллекцию, сколько раз они копировали ее и сколько раз из копий вашей коллекции совершались запросы API (подробнее см. на странице https://mng.bz/rVQg).

Стоит отдельно упомянуть *метрики тщеславия* (vanity metrics). К этой категории относятся метрики, которые не помогают понять эффективность продукта и не дают информации для его будущего улучшения.

Метрики вовлеченности

Метрики вовлеченности (engagement metrics) оценивают, сколько пользователей приобретают вашу документацию API и инструменты портала разработчика, когда они узнают о вашем API и понимают, что ваш API решает их задачу. И снова для оценки можно воспользоваться инструментами веб-аналитики. Примеры таких метрик:

- среднее время, проведенное на портале разработчика;
- *показатель отказов* (bounce rate) на портале разработчика, то есть доля посетителей, которые просматривают лендинг или портал разработчика и покидают сайт, не переходя на другие страницы.

Метрики привлечения

Метрики привлечения (acquisition metrics) измеряют количество разработчиков, впервые регистрирующихся для использования вашего API. После просмотра документации заинтересованные пользователи смогут опробовать ваш API, чтобы лучше понять, как он работает. Провайдеры API предлагают им

https://liveinternet.club

зарегистрироваться, чтобы подтвердить свой интерес и получить ключ API. Это позволяет им поработать с API в изолированной среде-песочнице. Провайдеры API также могут предоставлять пробные инструменты на базе веб-технологий на портале разработчика или коллекции Postman для упрощения задачи разработчика. Примеры метрик привлечения, которые отслеживаются продуктами API на этой стадии:

- количество регистраций новых пользователей;
- время, необходимое новому пользователю до первого вызова API, TTFC (Time to First Call) или TTFHW (Time to First Hello World);
- количество пользователей, опробующих ваши SDK или конкретную версию вашего API (принятие SDK и версий).

Метрики активации

Если после оценки вашего API разработчик понял, что API решает его задачу, и захотел использовать его в своем приложении, его следующий шаг — интеграция своего приложения с вашим API и развертывание его на продакшен. Метрики активации (activation metrics) отслеживают этот переход пользователей в категорию активных.

- В каждой организации существует свое определение активного пользователя API и соответствующая метрика. Она может быть предельно простой — например, количество пользователей, использующих API на продакшен (эта метрика отслеживается по использованию рабочего ключа API или токена) за определенный период времени. Обычно такие метрики называются DAU (Daily Active Users, *активные пользователи за день*), WAU (Weekly Active Users, *активные пользователи за неделю*) и MAU (Monthly Active Users, *активные пользователи за месяц*) в зависимости от интервала измерений. Так как пользователи-разработчики представляются ключами API, эти метрики также иногда обозначаются сокращениями DAT (Daily Active Tokens, активные токены за день), WAT (Weekly Active Tokens, активные токены за неделю) и MAT (Monthly Active Tokens, активные токены за месяц).
- Другая важная метрика — промежуток времени, необходимый пользователю для первого вызова API на продакшен. Метрика называется TTFT (Time to First Transaction, время до первой транзакции) или TFP (Time to First Production API Call, *время до первого вызова API на продакшен*).
- В некоторых организациях простого вызова API на продакшен недостаточно. Пользователь должен использовать API для завершения бизнес-транзакции (или нескольких транзакций), обладающих ощутимой ценностью. Они могут включать несколько вызовов API. Необходимое для этого время называется TTV (Time To Value, *время до получения ценности*). Например, для API интернет-магазина это может быть использование API для размещения заказов, превышающих заданный средний размер заказа.

https://liveinternet.club

Метрики удержания

Метрики удержания (retention metrics) показывают, какая часть потребителей API остается и использует продукт за определенный период времени. Это метрика лояльности, или приверженности пользователей продукту. Противоположностью удержания является *отток* — процент пользователей, которые перестают использовать API. Вот примеры таких метрик:

- *Повторное использование* (recurring usage) — метрика общего объема вызовов API за период времени (день, неделю, месяц). Анализ повторного использования для пользователя или группы (например, из одной компании) помогает выявить тенденции использования.
- *Удержание пользователей* (user retention) — метрика, оценивающая, насколько часто пользователи возвращаются для использования вашего API. Может определяться тем, сколько раз пользователь отправлял повторный запрос к заданному эндпоинту API в 7- или 30-дневном окне. Пользователей объединяют в группы (называемые *когортами*) по дате их первого запроса. Затем подсчитывают процент пользователей из каждой когорты, ежедневно отправляющих запросы за каждые 7 или 30 дней.

Метрики пользовательского опыта

Метрики пользовательского опыта (experience metrics) измеряют степень удовлетворенности клиентов общим опытом взаимодействия с вашим API. К таким метрикам относятся:

- *Общее количество потребителей API.* Потребителями могут быть отдельные пользователи или компании (несколько пользователей могут принадлежать одной компании). Ваша система управления учетными записями и метод аутентификации помогут связать используемый токен аутентификации (например, ключ API) с пользователем. Знание потребителей API поможет извлечь ценные сведения из их профилей, выявить тенденции и улучшить пользовательский опыт.
- *Ключевые потребители (top users) по объему использования API.* Это ваши самые успешные пользователи, и вы можете узнать много полезного из мониторинга паттернов использования и опросов этих пользователей.
- *Коэффициент конверсии.* Это доля пользователей (от их общего числа), которые «конвертируются», то есть достигают некоторой цели, определяемой бизнесом.

Из этого краткого обзора мы видим, что каждая категория включает несколько метрик (причем, как отмечалось выше, этот список не полон; вы даже можете разработать собственные нестандартные метрики). Чтобы использовать эти метрики, менеджеры продуктов API определяют, какие из них необходимо собирать на разных стадиях пути разработчика. Это помогает согласовать улучшение пользовательского опыта с достижением фокус-метрики со стороны бизнеса. На рис. 11.5 представлены точки возможного использования этих метрик на пути разработчика. Цель — определить не все возможные метрики API (такая работа

https://liveinternet.club

может быть слишком трудоемкой, и ее лучше избегать), а лишь небольшой их набор в каждой категории, чтобы управлять улучшением фокус-метрики для бизнеса.

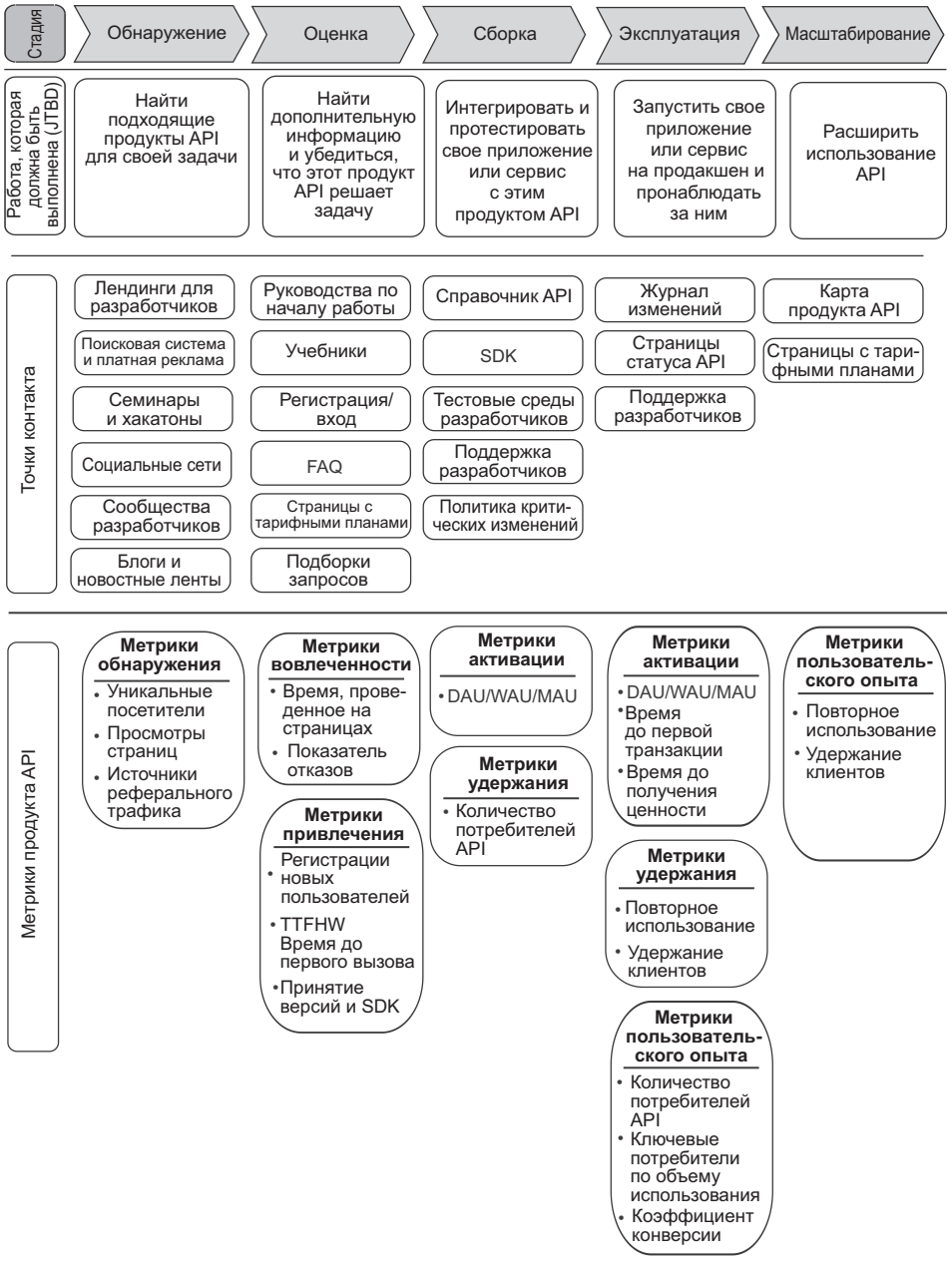


Рис. 11.5. Карта пути разработчика, дополненная категориями метрик продукта API

https://liveinternet.club

Мы обсудили, как аналитика продукта направляет цикл улучшения продукта и как сохранять метрики на разных стадиях пути разработчика. А теперь рассмотрим практический пример интеграции инструментов аналитики продуктов API в API-шлюз и сбора метрик продукта для пути разработчика.

11.1.4. Отслеживание метрик для API каталога продуктов

Вернемся к сценарию, описанному в разделе 11.1. Руководство поставило цель увеличить количество пользователей API каталога продукта на 10 % в следующем квартале, и вам поручили настроить метрики продуктов API для отслеживания текущей эффективности продукта и определения базового показателя для этого улучшения. Допустим, в этом сценарии вам поручено собирать метрики только для стадий оценки, сборки и эксплуатации на пути разработчика.

Сначала точно определите значение метрик для API каталога продуктов. Представьте, что команда API обсудила бизнес-цели и предложила следующие метрики продукта, которые должны отслеживаться на пути разработчика (рис. 11.6).

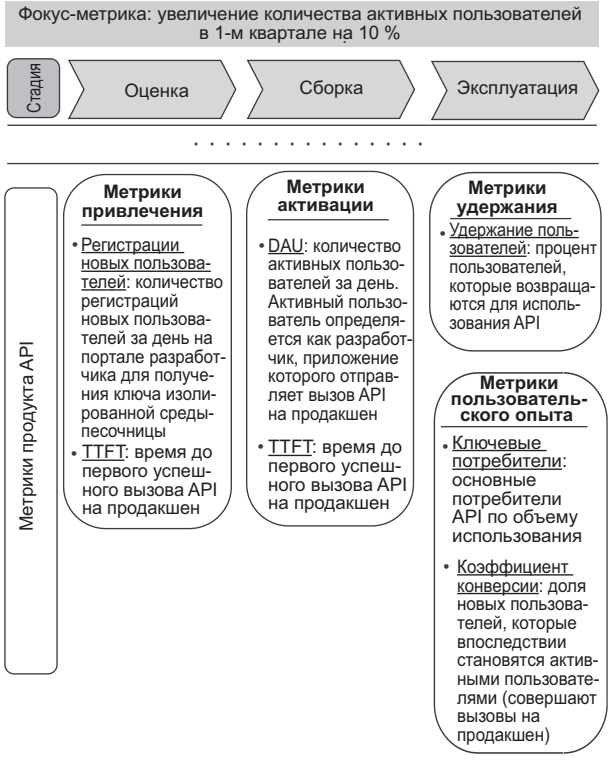


Рис. 11.6. Часть карты пути разработчика, показывающая, какие метрики команда API определила для управления улучшениями API каталога продуктов

https://liveinternet.club

Ключевой метрикой, которая влияет на фокус-метрику со стороны бизнеса, является DAU (активные пользователи за день) — повышение DAU должно приводить к повышению MAU. Обратите внимание на определение активных пользователей — это пользователи, отправляющие вызовы API на продакшен. Однако шесть других метрик важны для поддержания общей жизнеспособности продукта. Значения этих метрик можно определять с помощью инструментов аналитики продуктов API. Такие инструменты собирают и анализируют данные об опыте взаимодействия с API и использовании API, а также генерируют отчеты. Для нашего примера я покажу, как использовать один из таких инструментов — Moesif.

Почему именно Moesif?

На рынке представлено множество инструментов наблюдений и аналитики, как коммерческих, так и решений с открытым исходным кодом. Например, Prometheus и Grafana — инструменты с открытым кодом, которые обычно используются вместе для мониторинга и визуализации метрик. Кроме того, существуют такие коммерческие решения, как New Relic, Datadog, AppDynamics и др.

В этой главе я применяю Moesif, поскольку этот инструмент предоставляет первоклассную поддержку аналитики продукта API и включает набор предварительно определенных отчетов аналитики API, удобных в использовании. Кроме того, он превосходно интегрируется с API-шлюзом Kong, который используется в книге.

Сначала я покажу, как сохранить первую метрику привлечения — количество регистраций новых пользователей за день.

Регистрации новых пользователей

Эта метрика предназначена для отслеживания регистраций новых пользователей. Ваша гипотеза такова, что увеличение количества новых регистраций должно приводить к росту числа активных пользователей. Чтобы получать значения этой метрики, необходимо отправлять в Moesif событие при каждой регистрации нового пользователя. Также необходимо создать в Moesif диаграмму временно́го ряда для визуализации и распространения для всей команды данных о тенденциях регистраций за день.

Одна из разновидностей событий, сохраняемых Moesif, — *пользовательское действие*. Это некая операция, выполняемая пользователем в пользовательском интерфейсе, — например, отправка формы или щелчок на кнопке. Пользовательские действия идеально подходят для сбора такой информации, как регистрация учетной записи разработчика на портале и получение ключа API. Moesif предоставляет коллекцию средств интеграции на стороне клиента, написанных на JavaScript, которые можно встроить в свой портал для упрощения задачи.

https://liveinternet.club

Чтобы не усложнять пример, я подготовил файл рабочего потока GitHub Action `.github/workflows/analytics.completed` для моделирования действия регистрации. Рабочий поток использует скрипт `simulate-signups.js` для моделирования регистрации и получения ключей API пользователями. Он также отправляет в Moesif пользовательское действие Sign-Up через Moesif API (так же, как это будет делать интеграционный код клиента). Затем скрипт `simulate-api-calls.js` используется для моделирования вызовов пользователей к изолированной среде (с базовым путем `/sandbox`) и API на продакшен (с базовым путем `/api`) в API-шлюзе после их регистрации (по истечении заданного периода времени). Благодаря интеграции Moesif с этим экземпляром Kong шлюз отправляет Moesif событие API `Call` при каждом получении запроса API. Схема процесса показана на рис. 11.7.

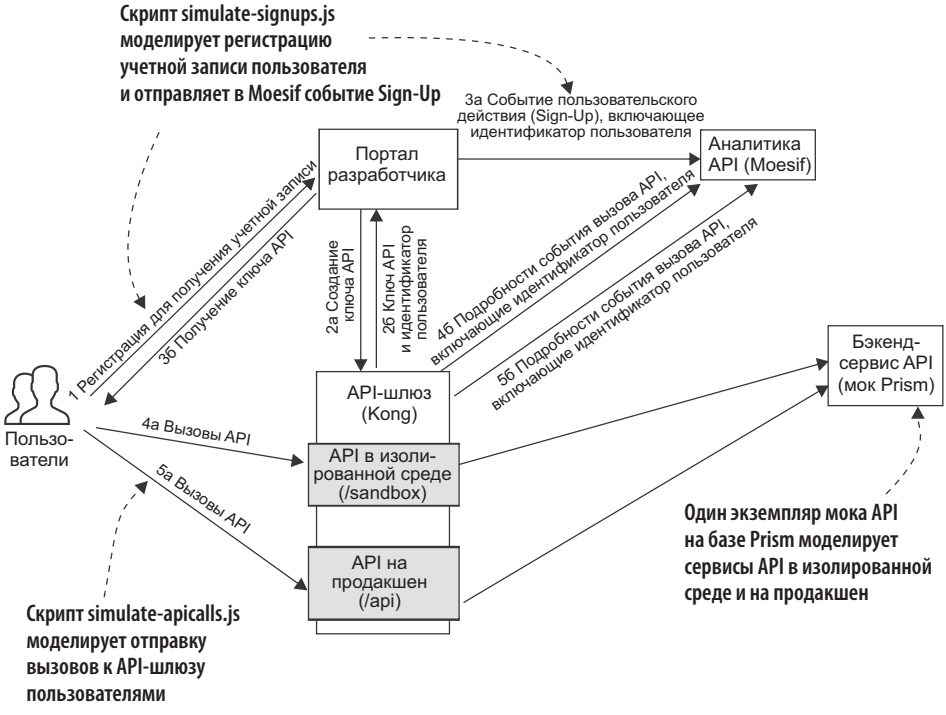


Рис. 11.7. Файл рабочего потока моделирует регистрацию пользователя на портале разработчика и вызовы API к шлюзу. Он отправляет в Moesif ассоциированные события пользовательских действий и события вызовов API

Чтобы использовать файл рабочего потока, сначала заблокируйте все активные рабочие потоки в каталоге `.github/workflows` — например, заменив их суффикс `.yaml` на `.disabled`. Затем переименуйте файл рабочего потока `.github/workflows/analytics.completed` в `.github/workflows/analytics.yaml`

https://liveinternet.club

и сохраните изменения. GitHub Actions обнаруживает файл рабочего потока и выполняет его.

ПРИМЕЧАНИЕ Файл рабочего потока должен выполниться, но, если у вас возникнут проблемы, убедитесь, что у следующих скриптов есть разрешения на выполнение: `chapter11/stop_kong.sh`, `chapter11/start_kong.sh`, `chapter11/simulate-signups.js`, `chapter11/configure-routes.sh`, `chapter11/restart_kong.sh` и `chapter11/simulate-apicalls.js`. (В системах семейства Unix можно воспользоваться командой `chmod +x <имя_файла>`.) Скрипт моделирования в этом примере выполняется в пайплайне примерно 15 минут и выводит значения TTFHW и TTFT в минутах. В реальном сценарии эти значения будут измеряться в днях, неделях или месяцах.

Теперь подключитесь к Moesif и создайте дашборд для просмотра зарегистрированных пользователей и анализа тенденций. Сначала создайте в Moesif дашборд для сохранения созданных отчетов. Выполните команду New ▸ Dashboard и введите имя `ProductCatalogDashboard`. Затем кликните на кнопке Add, чтобы сохранить созданный дашборд (рис. 11.8).



Рис. 11.8. Создание дашборда для аналитического отчета

Создайте отчет, содержащий все события, которые вы отправляете в Moesif. Выполните команду New ▸ Metrics ▸ Live Event Log. В отчете выводится поток событий, полученный Moesif, включая идентификатор пользователя и время события

https://liveinternet.club

(рис. 11.9). Сохраните отчет с именем `Event Log`, кликнув на кнопке Save в левой верхней части страницы. Если отчет не появился в разделе ProductCatalogDashboard левого меню, перезагрузите страницу.

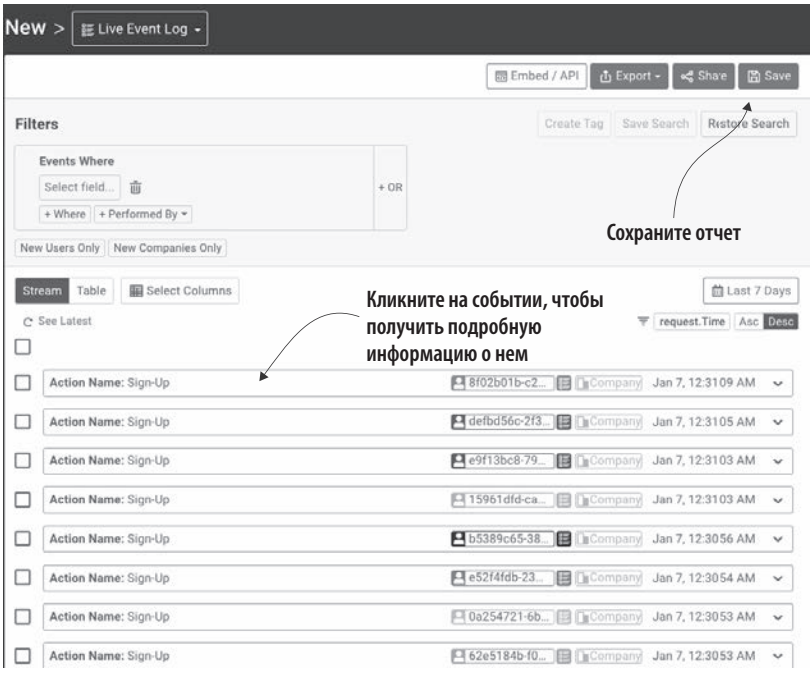


Рис. 11.9. Поток событий Moesif

Создайте отчет с информацией о регистрациях командой New ▸ Metrics ▸ Time Series. В подразделе Events Where раздела Filters создайте фильтр Action Name ▸ Is ▸ Sign-Up (рис. 11.10).

Кликните на крайней левой кнопке, чтобы данные отобразились на столбчатой диаграмме (рис. 11.11). В отчете приводятся тенденции количества регистраций за несколько последних дней; в частности, видно, что в день выполнения скрипта зарегистрировалось восемь пользователей.

Сохраните отчет в разделе ProductCatalogDashboard под именем `Daily New User Sign-ups`.

Создайте отчет для вывода подробной информации о зарегистрировавшихся пользователях. В Moesif такие отчеты относятся к категории Lookup. Выполните команду New ▸ Users ▸ Lookup. В фильтре Users Where выберите вариант Created ▸ Is ▸ After 1 Day Ago, как показано на рис. 11.12.

https://liveinternet.club

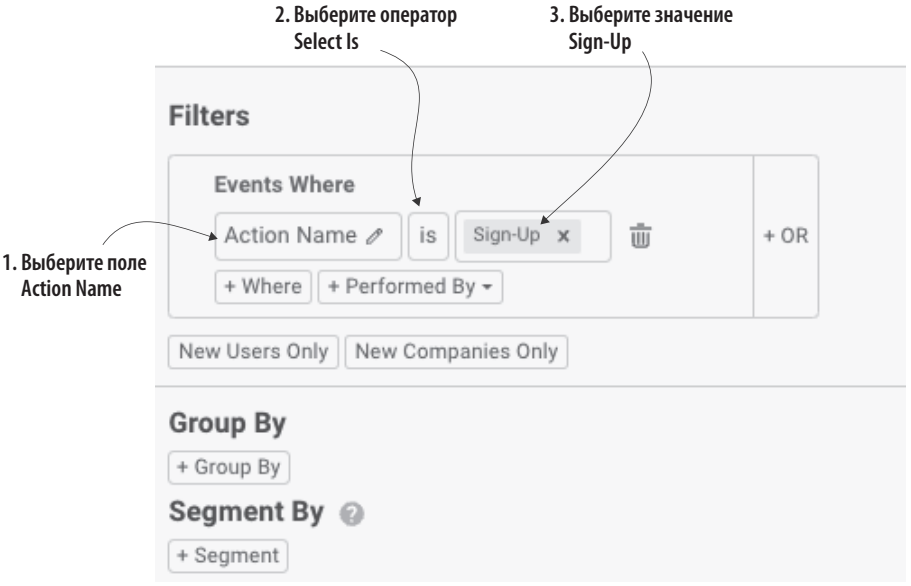


Рис. 11.10. Создание отчета временного ряда для просмотра информации о регистрации новых пользователей в Moesif

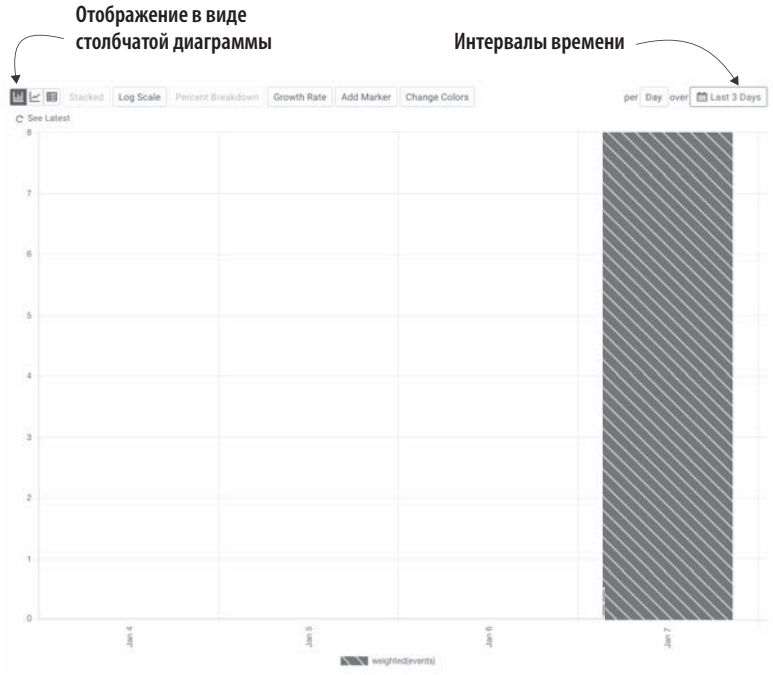


Рис. 11.11. Столбчатая диаграмма с тенденциями количества регистраций за последние три дня

https://liveinternet.club

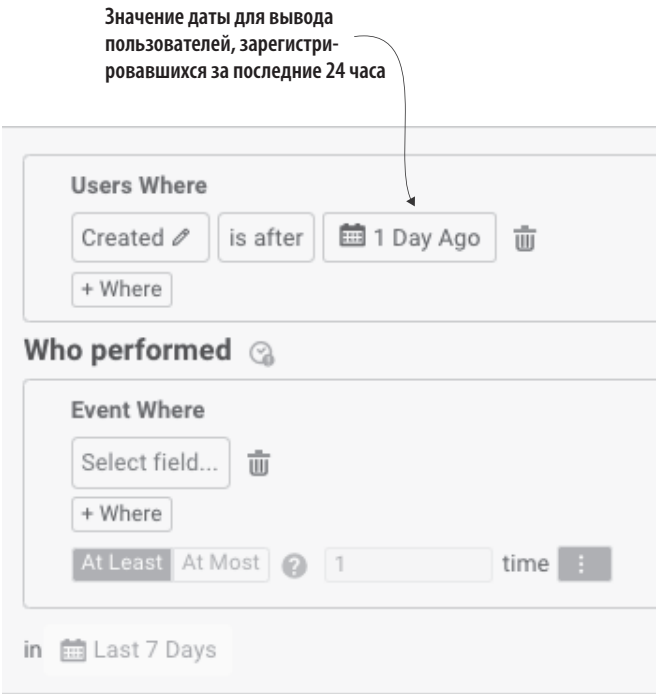


Рис. 11.12. Параметры отчета для вывода информации о недавно зарегистрировавшихся пользователях

Генерируется отчет с информацией о восьми зарегистрировавшихся пользователях (рис. 11.13). Сохраните представление в `ProductCatalogDashboard` под именем `New User Sign-ups in Last 24`, кликнув на кнопке Save в правом верхнем углу.

Итак, вы узнали, как получить информацию о последних регистрациях. Теперь посмотрим, как получить другую метрику привлечения — TTFHW.

TTFHW

Moesif вычисляет TTFHW (time to first Hello world) как разность между временем создания пользователя на платформе Moesif (в данном случае — регистрации пользователя) и временем совершения первого вызова API. Как говорилось выше, у запущенного экземпляра Kong установлен плагин Moesif Kong. Моделирующий рабочий поток совершает небольшое количество случайных запросов API в изолированной среде (с путем `/sandbox`) для шести из восьми зарегистрированных пользователей. Иначе говоря, два пользователя зарегистрировались, но не совершили ни одного вызова API. Чтобы просмотреть метрику TTFHW в Moesif, перейдите к созданному вами дашборду `New User Sign-ups in the Last 24 Hours`. Найдите столбец `ttfhw_s`. Вы увидите, что

https://liveinternet.club

он заполнен значением TTFHW (рис. 11.14). Так как запросы API от двух пользователей не отправлялись, значение TTFHW для них отсутствует и выводится в виде N/A (недоступно).

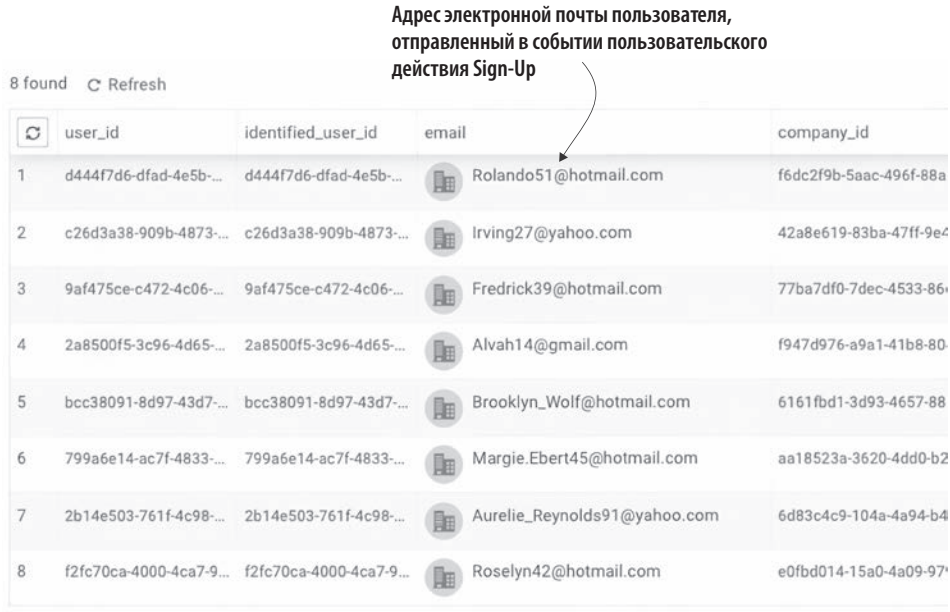


Рис. 11.13. Подробная информация о недавно зарегистрировавшихся пользователях

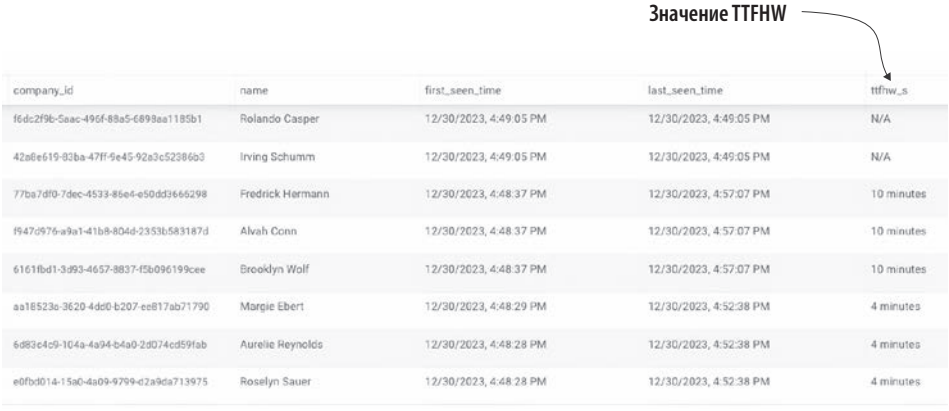


Рис. 11.14. Значение TTFHW для отчета с данными недавно зарегистрировавшихся пользователей

https://liveinternet.club

Вы можете просмотреть значения TTFHW для отдельных пользователей. А если нужно увидеть среднюю метрику TTFHW для всех зарегистрированных пользователей? Для этого можно создать *пользовательскую воронку*, которая показывает, какой процент зарегистрированных пользователей отправили запрос API и какое среднее значение имела метрика TTFHW. Анализ пользовательской воронки дает возможность наглядно представить и понять поведение пользователей и возможные препятствия на их пути.

Создайте пользовательскую воронку TTFHW. Выполните команду New ▸ Users ▸ Funnel. Установите параметры стадии 1 (Stage 1) с именем Signed Up с событиями Action Name ▸ Is ▸ Sign-Up. Установите параметры стадии 2 (Stage 2) с именем First API Request Event Type ▸ Is ▸ API Call и request.segments.0 ▸ Is ▸ Sandbox. На рис. 11.15 показано, как должна выглядеть конфигурация пользовательской воронки.

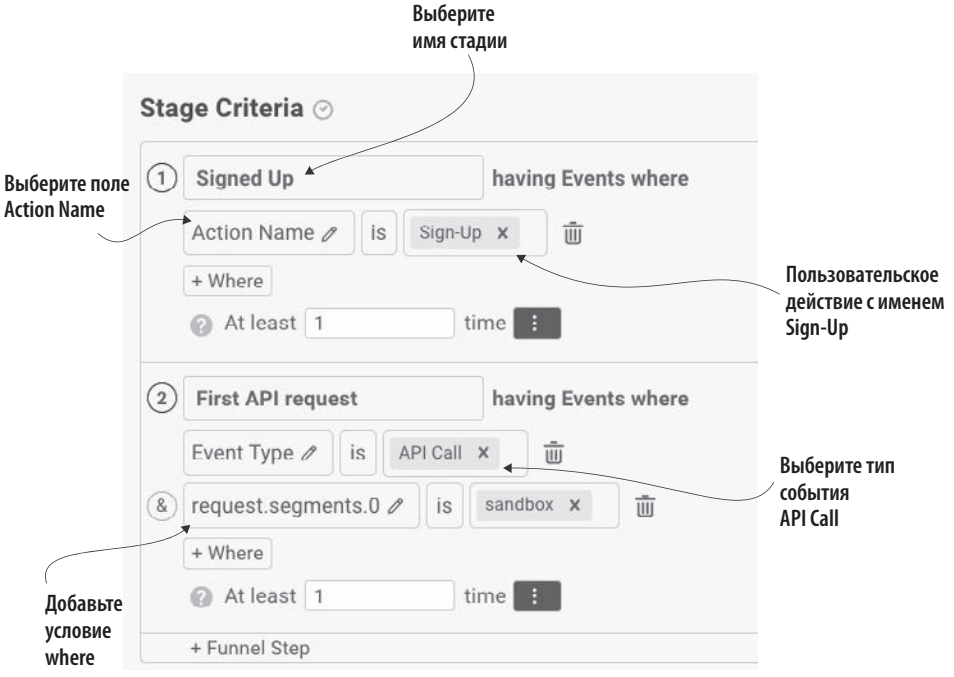


Рис. 11.15. Настройка параметров для анализа воронки TTFHW

С установленными параметрами диаграмма анализа воронки выглядит, как показано на рис. 11.16. (Учтите, что появление новых событий на диаграмме воронки может занять до 10 минут.) В моем примере запуска на рис. 11.16 видно, что только 75 % пользователей (6 из 8 — напомню, что в нашей модели не все пользователи отправили запрос после регистрации) перешли к вызовам API

https://liveinternet.club

после получения ключа API. Кроме того, среднее время, необходимое для таких запросов, составило 6 минут 19 секунд.

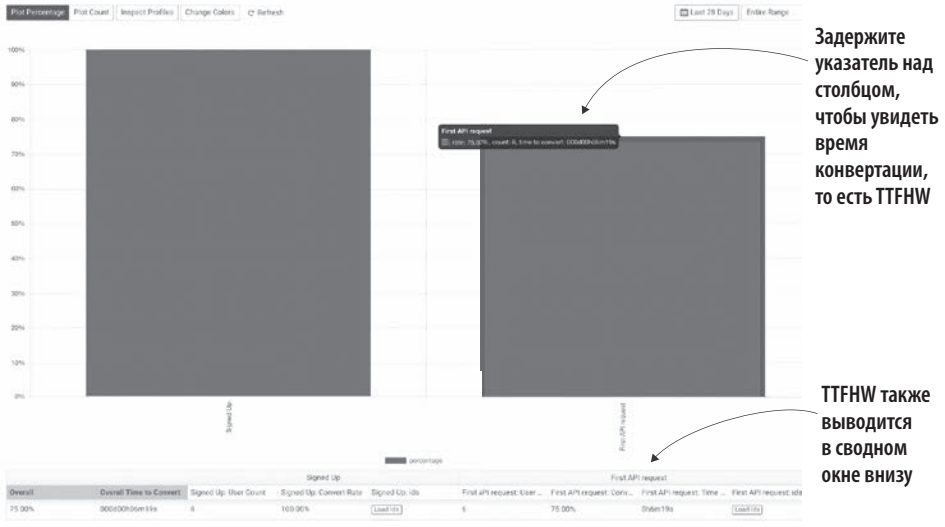


Рис. 11.16. Среднее значение TTFHW для группы из шести пользователей, которые совершили отправку запросов API после регистрации, составляет 6 минут 19 секунд

Сохраните представление под именем `User Funnel Analysis` в разделе `ProductCatalogDashboard`. Итак, я показал, как построить представления для двух метрик привлечения — количества регистраций новых пользователей и TTFHW. Теперь посмотрим, как создать представления для следующей стадии пути разработчика — стадии активации.

DAU (количество активных пользователей за день)

Вспомните, что команда определяет активного пользователя как того, кто совершает вызов API в продакшен-среде. В нашем примере вызовы к продакшен-среде имеют базовый путь `/api`. Чтобы измерить количество активных пользователей за день, следует создать диаграмму временного ряда с уникальными пользователями, отправляющими вызовы к URL на продакшен. Скрипт симуляции отправляет запросы к API на продакшен только для трех пользователей.

Перейдите к своей учетной записи Moesif, чтобы создать представление для DAU. Выполните команду New ▸ Metrics ▸ Time Series. Создайте фильтр Event Type ▸ Is ▸ API Call и Requests ▸ URI Segment ▸ 0 (то есть `request.segment.0` содержит `Api`). В поле Metrics измените метрику Event Count на Uniques ▸ Unique Users. На рис. 11.17 показаны установленные параметры.

https://liveinternet.club

На этой диаграмме представлено количество уникальных пользователей, которые отправили запросы на продакшен. Кликните на значке гистограммы, чтобы вывести данные именно в таком виде. Как и предполагалось, вы увидите трех активных пользователей. На рис. 11.18 показано, как это выглядит на моем компьютере.

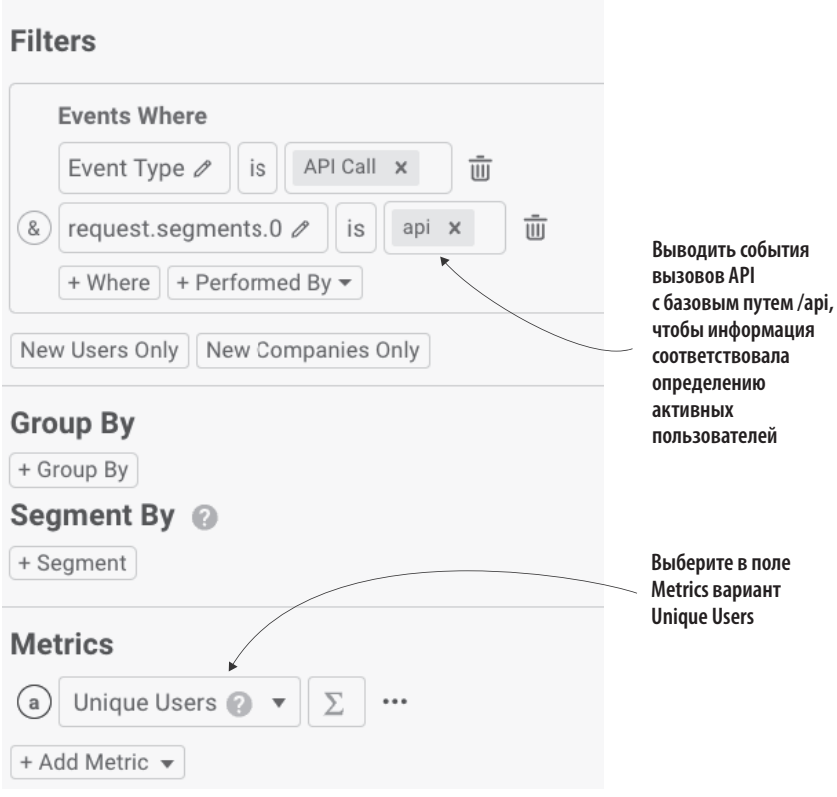


Рис. 11.17. Параметры DAU для вывода активных пользователей, отправляющих вызовы API по базовому пути /api

Сохраните представление в `ProductCatalogDashboard` с именем `Daily Active Users (DAU)`.

Вы узнали, как просмотреть статистику активных пользователей. Теперь мы наглядно представим вторую метрику активации — время до первой транзакции на продакшен.

https://liveinternet.club

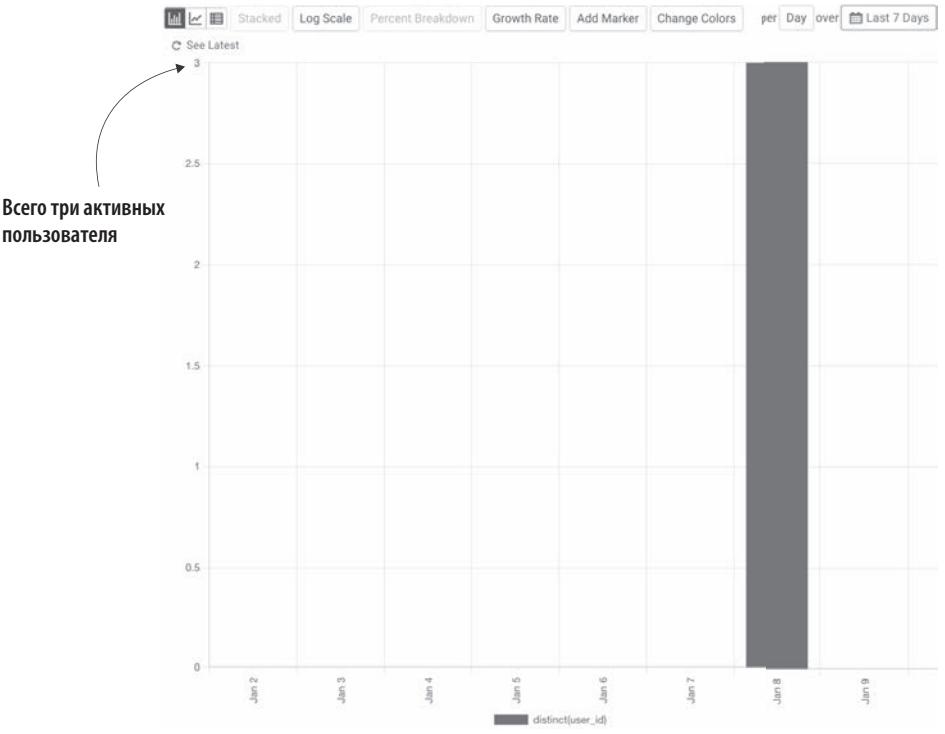


Рис. 11.18. Пример диаграммы временного ряда DAU

TTFT

Вспомните, что для нашего примера команда определила TTFT (Time To First Transaction) как время, необходимое для совершения первого вызова API на продакшен. Для вывода этой информации можно обновить представление анализа воронки.

Перейдите к отчету `User Funnel Analysis`, созданному ранее. Добавьте новый шаг воронки, чтобы вывести информацию о вызовах на продакшен, кликнув на кнопке + Funnel Step. Присвойте новому шагу имя `First Production Call`. Задайте для него параметр request.segment.0 ▸ Is ▸ Api (рис. 11.19) и сохраните представление.

https://liveinternet.club

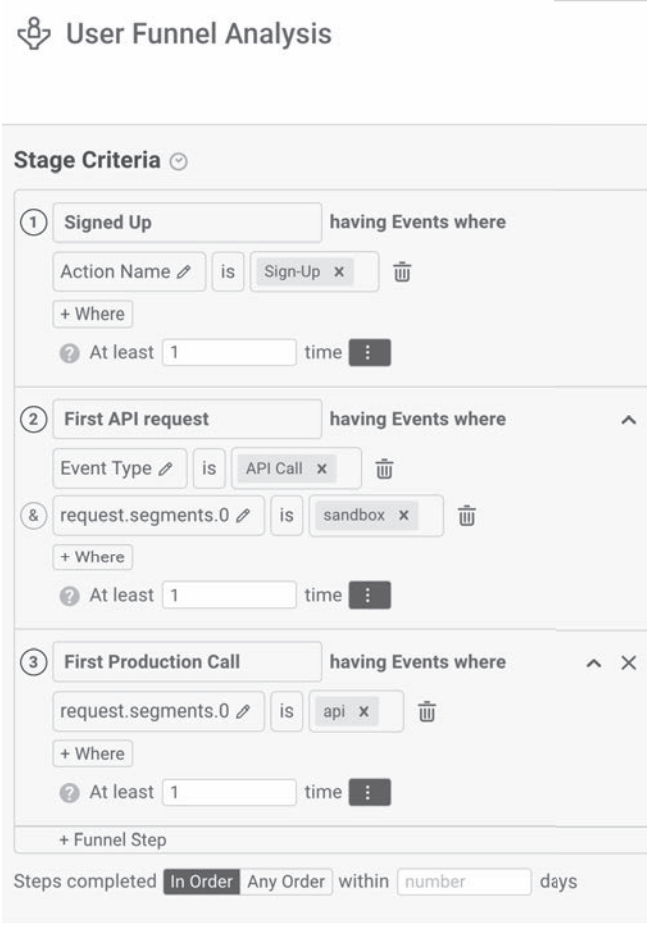


Рис. 11.19. Добавление дополнительного параметра First Production Call в отчет User Funnel Analysis, чтобы в нем отображалось значение TTFT

На моем компьютере получилась диаграмма, показанная на рис. 11.20. Наводя указатель мыши на столбец First Production Call, видим, что TTFT составляет 12 минут.

Теперь настроим метрики удержания пользователей для стадии удержания.

Удержание пользователей

Эта метрика показывает, какой процент пользователей возвращаются к использованию выбранного API с течением времени. Чтобы настроить отчет о ежедневном удержании пользователей, выполните команду New ▸ Users ▸ Retention. В параметрах событий (Event Criteria) выберите для First Event и Returning Event

https://liveinternet.club

конфигурацию Event Type ▸ Is ▸ API Call. На рис. 11.21 показаны установленные параметры.

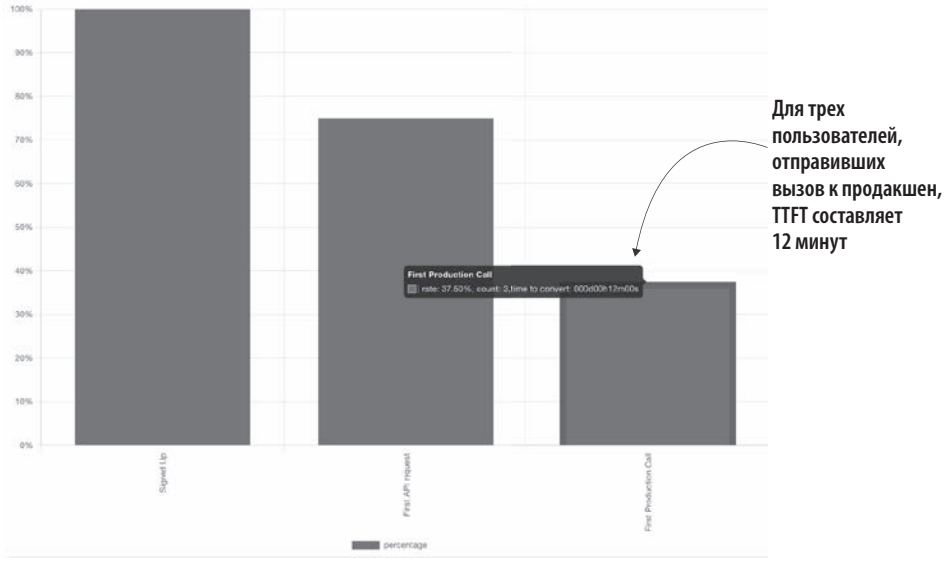


Рис. 11.20. Обновление диаграммы User Funnel Analysis для отображения TTFT

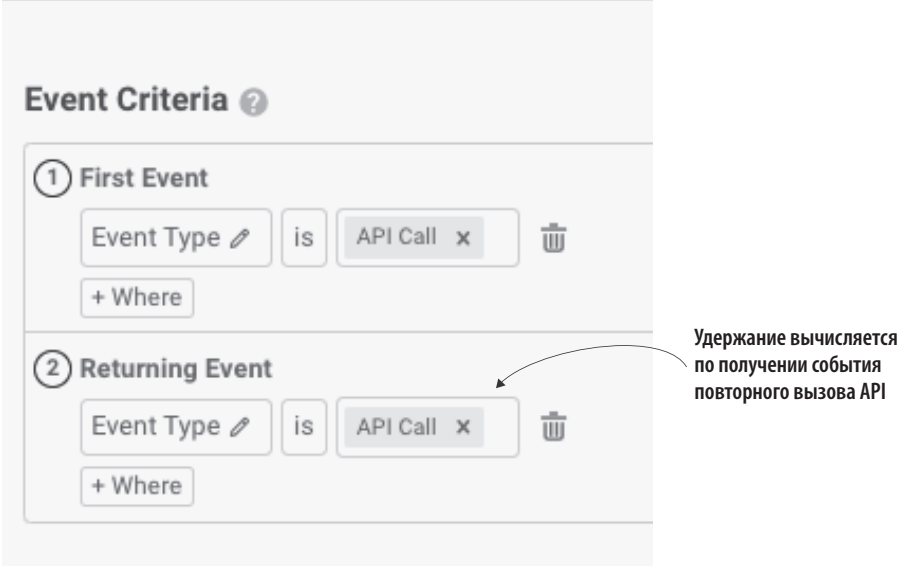


Рис. 11.21. Настройка параметров для диаграммы удержания пользователей

https://liveinternet.club

На моем компьютере выводится диаграмма, показанная на рис. 11.22. Так как после первого дня не было совершено ни одного повторного вызова, частота удержания в первый день падает до нуля (день получения первого запроса от пользователя считается нулевым днем).

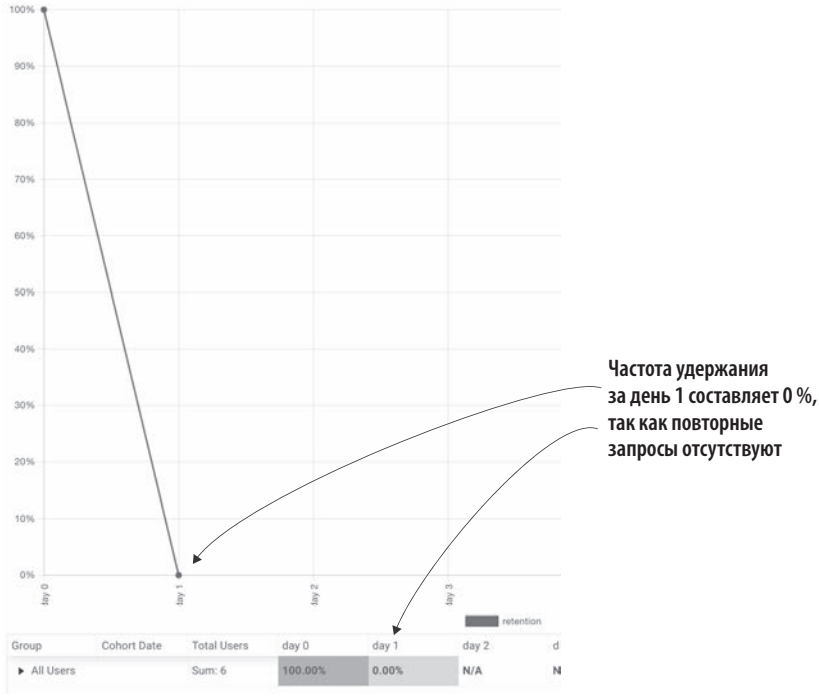


Рис. 11.22. Диаграмма удержания пользователей

Сохраните это представление в `ProductCatalogDashboard` с именем `Daily User Retention`.

> **СОВЕТ** Чтобы увидеть пример диаграммы удержания пользователей с данными за несколько дней, взгляните на одно из демонстрационных приложений. В левой верхней части экрана переключите свою учетную запись Moesif Demos ▸ SaaS. Затем на появившемся дашборде Product — E-sign API кликните на отчете Daily User Retention.

Теперь перейдем к метрикам пользовательского опыта — коэффициенту конверсии и ключевым пользователям API.

Коэффициент конверсии

Команда определила коэффициент конверсии как процент пользователей, которые сохраняют активность, то есть продолжают отправлять вызовы API на продакшен. Чтобы увидеть это значение, вернитесь к представлению User Funnel

https://liveinternet.club

Analysis на рис. 11.23. Как видно из таблицы в нижней части рисунка, коэффициент конверсии составляет 37,5 %. Иначе говоря, 37,5 % зарегистрированных пользователей продолжают отправлять вызовы API на продакшен.

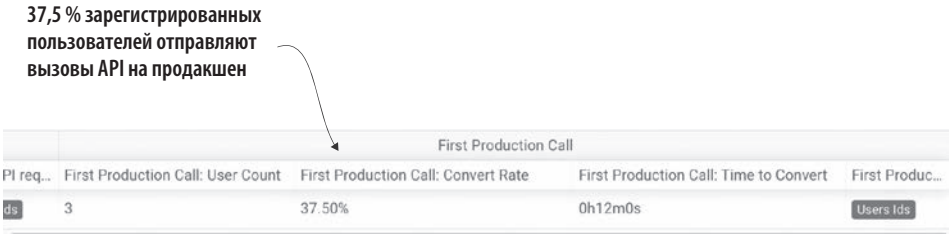


Рис. 11.23. Отчет User Funnel Analysis показывает коэффициент конверсии пользователей, отправляющих вызовы API на продакшен

> **ПРИМЕЧАНИЕ** На практике выбор критерия конверсии зависит от бизнес-модели и стратегии монетизации API. За обсуждением различных стратегий монетизации, тарификации и упаковки для API обращайтесь к книге Майка Амундсена (Mike Amundsen) и Деррика Гриллинга (Derric Grilling) «API Strategy for Decision Makers: Growing Your Product Through Customer Observability» (2022, O'Reilly Media).

Перейдем к последней метрике — ключевым пользователям API.

Ключевые пользователи API на продакшен

Чтобы получить список ключевых пользователей для каждой среды, создайте диаграмму сегментации. Выполните команду New ▸ Metrics ▸ Segmentation. Чтобы учесть только вызовы в продакшен-среде, задайте фильтр Event Type ▸ Is ▸ API Call и request.segments.0 ▸ Is ▸ Api. Чтобы сгруппировать запросы по пользователям, обновите поле Group By значением User.User Name и настройте его для вывода только первых пяти значений (в данном случае имен пользователей). Введите в поле Metrics значение Event Count. Параметры должны выглядеть как на рис. 11.24.

И хотя я настроил показ пяти ключевых пользователей, запросы API к продакшен поступают только от трех пользователей. Как видно на рис. 11.25, самые активные пользователи — Roselyn Sauer (18 вызовов API) и Margie Ebert (15 вызовов).

Сохраните эту диаграмму с именем `Top Users in Production`. Теперь вы настроили семь метрик для отслеживания пути разработчика и предоставили своей команде и менеджменту информацию для измерения эффективности продукта.

В этом упражнении мы использовали лишь несколько точек данных. Moesif содержит подборку демонстрационных приложений, с помощью которых вы можете познакомиться с другими типами аналитических диаграмм. Чтобы переключиться на них и просмотреть их предварительно настроенные дашборды, кликните на своем имени пользователя в левой верхней части экрана и выберите вариант Moesif Demos в раскрывающемся списке.

https://liveinternet.club

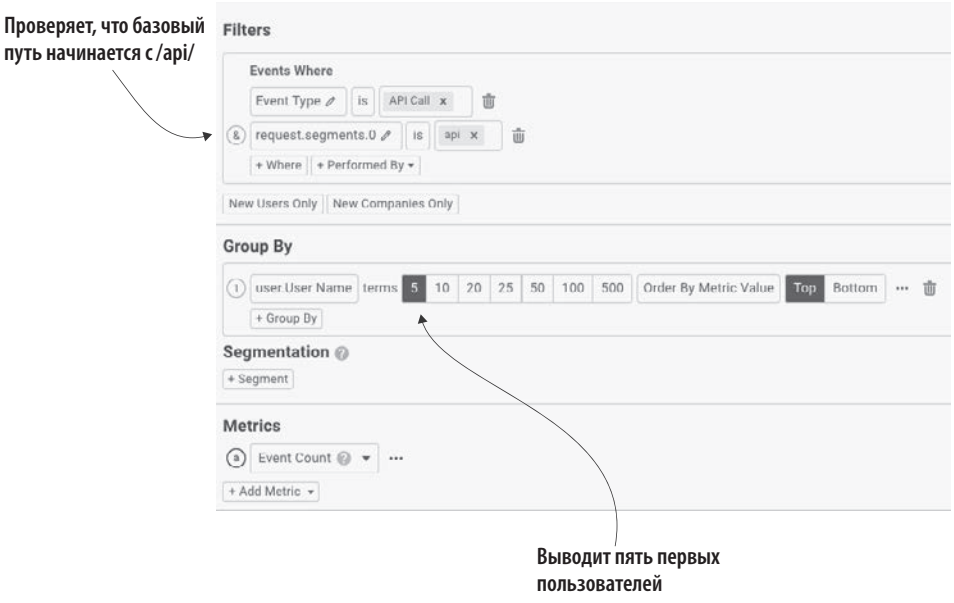


Рис. 11.24. Параметры вывода пяти первых пользователей в продакшен-среде по количеству отправленных ими вызовов API

Мы рассмотрели метрики эффективности продукта. Перейдем к обсуждению метрик технической производительности для обнаружения и диагностики технических проблем с API.

11.2. Метрики мониторинга API

Ваш API каталога продуктов работает на продакшен. Представьте, что у него возникли серьезные проблемы с производительностью, он возвращает непредвиденные ошибки или стал работать очень медленно. Как команда узнает об этом? Что она должна проверить в первую очередь, чтобы определить причину происходящего? Для этого необходимо настроить мониторинг API для получения метрик технической производительности.

При обсуждении практик DevOps в главе 1 вы узнали, что суть мониторинга заключается в использовании предварительно определенных метрик, дашбордов и уведомлений, позволяющих командам понять внутреннее состояние своих систем. Мониторинг предоставляет командам информацию о работоспособности API, используя данные событий для диагностики и исправления проблем.

Существует множество технических метрик, касающихся производительности, интенсивности использования и надежности, которые можно собрать для приложений и сервисов. Эти метрики делятся на две категории — инфраструктурные метрики и метрики уровня сервисов.

https://liveinternet.club

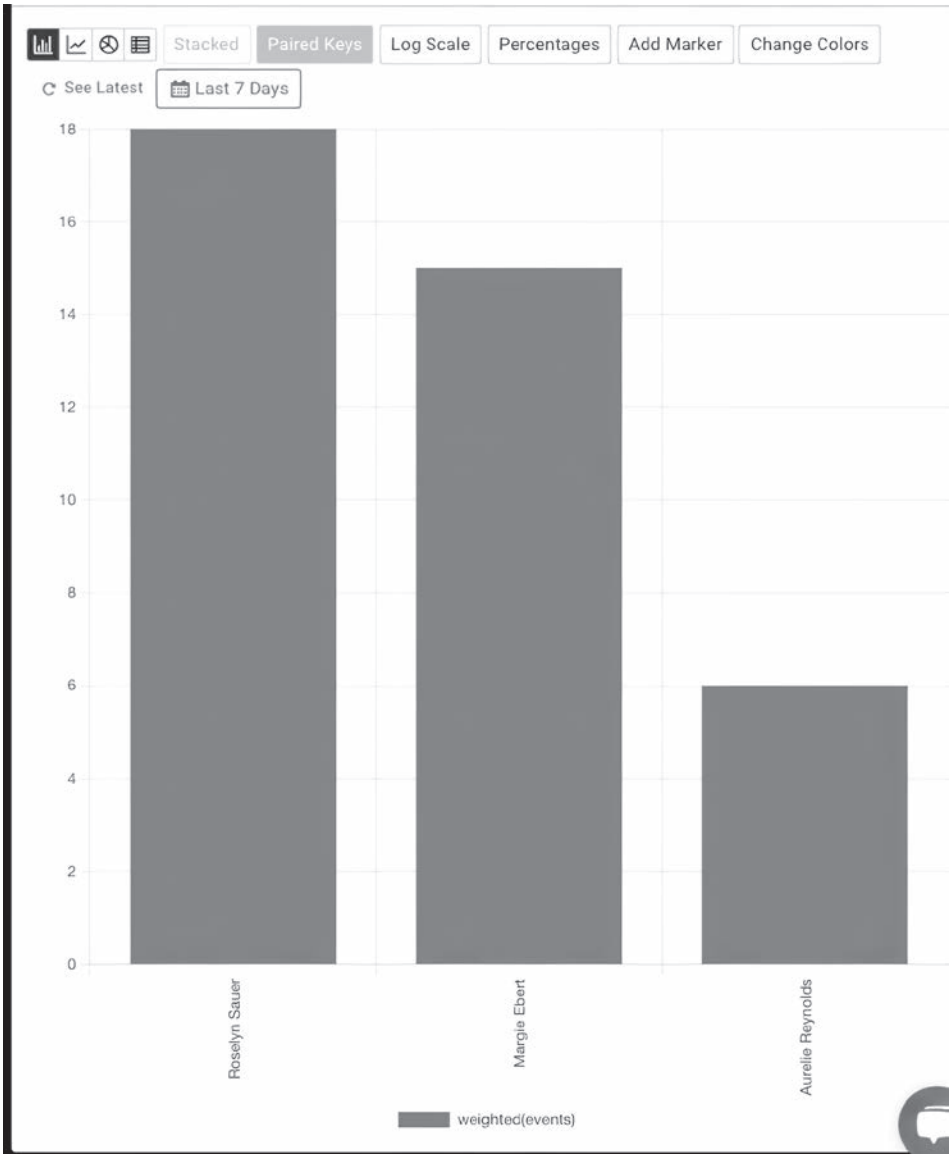


Рис. 11.25. Ключевые пользователи API

Инфраструктурные метрики охватывают аппаратные и программные инфраструктурные компоненты, на которых выполняются приложения и сервисы. К их числу относятся метрики загрузки процессора, памяти, дискового ввода/вывода, очередей, загрузки сети, подключений к базам данных и т. д. Метрики уровня сервисов относятся к работоспособности отдельных приложений или

https://liveinternet.club

сервисов: это задержка, частота ошибок, пропускная способность системы, время работоспособности и т. д.

Тем не менее без четкой стратегии в большом количестве метрик можно запутаться. Чтобы сосредоточиться на важных метриках, можно воспользоваться популярными формулами мониторинга SRE (Site Reliability Engineering). Они включают «четыре золотых сигнала», метод RED (частота запросов, частота ошибок, продолжительность — Request rate, Error rate, Duration) и метод USE. В этой книге рассматривается метод RED, так как он главным образом ориентирован на микросервисные архитектуры.

Другие формулы мониторинга SRE

Хотя в следующем разделе рассматривается метод RED, полезно иметь представление о других формулах мониторинга SRE. В частности, для физических компонентов серверов (например, процессоров и дисков) часто применяется метод USE, который включает следующие метрики:

- *загруженность* (Utilization) — среднее время занятости ресурса;
- *насыщение* (Saturation) — объем находящейся в очереди работы, которую ресурс не может обслужить;
- *ошибки* (Errors) — количество ошибок ресурса.

Этот метод более подробно обсуждается в статье Брендана Грегга (Brendan Gregg) «The USE Method» (www.brendangregg.com/usemethod.html).

Еще одна популярная формула — «четыре золотых сигнала», также называемая методом LETS. В ней измеряются следующие метрики для систем, обращенных к пользователю:

- *задержка* (Latency) — время обслуживания запроса;
- *ошибки* (Errors) — частота неудачных запросов (например, количество неудачных запросов HTTP в секунду);
- *трафик* (Traffic) — объем нагрузки на систему (например, количество запросов HTTP в секунду);
- *насыщение* (Saturation) — степень загруженности или ограниченности сервиса относительно его текущей нагрузки. Насыщение позволяет определить, какой ресурс является ключевым ограничивающим фактором (то есть это ресурс, который очень сложно увеличить). Например, это может быть процессор или пропускная способность сети.

Метод «четырех золотых сигналов» был разработан в Google. Дополнительную информацию можно найти в публикации «Site Reliability Engineering: How Google Runs Production Systems» (O'Reilly, 2021).

Разновидностью метода «четырех золотых сигналов» является метод STELA, который добавляет в формулу LETS еще одну метрику — доступность (Availability). Доступность определяет степень работоспособности и возможности использования сервиса. Она часто измеряется в «девятках» — например, система с доступностью «три девятки» (99,9 %) находится в нерабочем состоянии не более 8,76 часа за год.

https://liveinternet.club

11.2.1. Метрики RED

Метод RED определяет философию мониторинга для микросервисных архитектур. Он был разработан в Weaveworks, а центральное место в нем занимают три ключевые метрики для каждого сервиса, которые описаны в следующих подразделах.

Частота запросов

Эта метрика, также называемая пропускной способностью, определяет количество запросов, которые может обработать API за определенное время. Ее можно измерять для коротких промежутков времени (например, за секунды или минуты) либо для более длительных интервалов (например, за неделю или месяц). Она помогает понять тенденции и закономерности использования API.

Частота ошибок

Метрика определяется как количество неудачных запросов за определенное время (например, за секунду или за минуту). В нашем примере должны отслеживаться ошибки HTTP 4xx (ошибки пользователя) и ошибки 5xx (ошибки на стороне сервера). Очень важно иметь информацию о контексте проблемы. Например, большое количество ошибок 401 Unauthorized (не авторизован) для конкретного географического региона (с выводом ошибок на географической карте) может указывать на действия злоумышленников или ботов. Информация из журналов приложения поможет понять, какой компонент инфраструктуры или приложения вызывает ошибку 5xx и почему это происходит. Также следует создать оповещения для команды API о превышении заданного порога частоты ошибок.

Продолжительность/задержка запросов

Это время завершения запроса API, то есть время от момента создания запроса пользователем до получения ответа. Эта метрика важна, поскольку высокая задержка замедляет обслуживание и ухудшает пользовательский опыт.

Задержка запросов обычно измеряется на уровне шлюзов в микросервисной архитектуре. Однако вывод только средней задержки по всем эндпоинтам может скрыть эндпоинты с низкой производительностью. Следовательно, полезно выводить распределение задержек по эндпоинтам, а также представлять данные в процентилях, как на рис. 11.26.

ПРИМЕЧАНИЕ Процентиль — механизм отделения определенного процента значений в наборе данных от других значений. Например, если 90-й процентиль задержек API ниже 250 мс в наборе задержек API составляет 250 мс, это означает, что 90 % запросов API выполняется за 250 мс и быстрее, а 10 % запросов выполняются более чем за 250 мс.

https://liveinternet.club

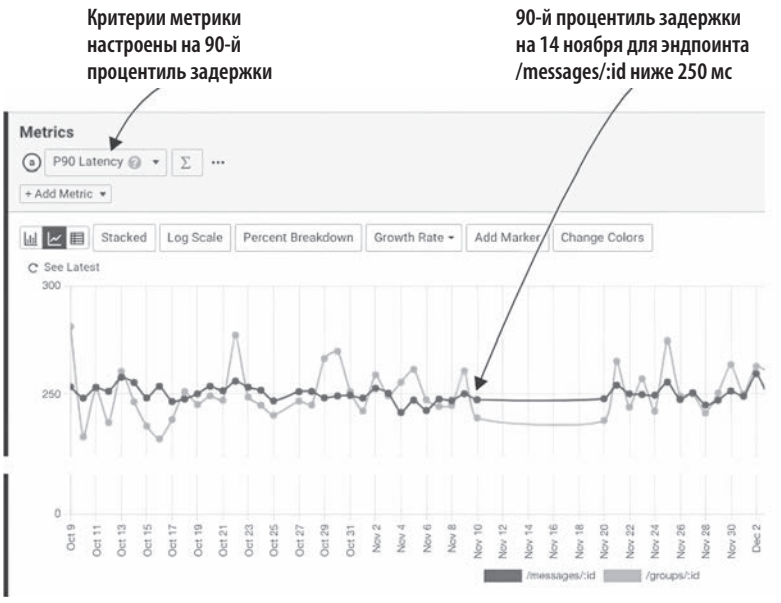


Рис. 11.26. Диаграмма задержки в Moesif, отслеживающая 90-й процентиль задержки от двух эндпоинтов, /messages/:id и /groups/:id

Измерять задержку API полезно от лица пользователей из различных географических регионов. Это гарантирует охват всех задействованных компонентов, включая время подключения TCP, согласование SSL, поиск DNS и время обработки на бэкенд-сервере. Существуют различные инструменты мониторинга API (например, APIMetrics (https://apimetrics.io)), предоставляющие возможность измерения задержки от разных облачных дата-центров со всего мира.

Мы рассмотрели метод RED как удобную формулу для измерения технической производительности API. Но прежде чем завершать обсуждение метрик, я хочу упомянуть одно важное ограничение, которое необходимо устанавливать для API, — ограничение частоты.

11.2.2. Ограничение частоты

Ограничение частоты (rate limit) — механизм ограничения количества запросов, с которыми потребитель может обращаться к API. Как объяснялось в главе 9, ограничения частоты помогают защитить API от избыточного трафика и некоторых видов атак. Ограничение частоты гарантирует, что все пользователи получат справедливый доступ к API, а их работа будет последовательной и предсказуемой. В своей книге «API Traffic Management» (O'Reilly Media, 2019) Майк Амундсен (Mike Amundsen) упоминает о том, что ограничения частоты иногда называются *точками насыщения*, чтобы показать, что это частота успешной обработки запросов системой.

https://liveinternet.club

В зависимости от их продолжительности ограничения можно разделить на краткосрочные и долгосрочные. Краткосрочные устанавливаются для защиты серверов API от перегрузки пиками трафика. (Некоторые шлюзы — например, Apigee — используют для этого термин *«блокировка выбросов»*.) Они измеряются подсчетом количества запросов от конкретного IP-адреса или ключа API за короткое время, например за секунду или за минуту. Пример диаграммы временного ряда с количеством запросов в минуту и маркером ограничения частоты показан на рис. 11.27. Эта диаграмма поможет команде API быстро понять, было ли превышено ограничение частоты. Если у API-шлюза включена функциональность ограничения частоты, то запросы после превышения частоты получат ответ с кодом состояния 429 Too Many Requests (слишком много запросов).

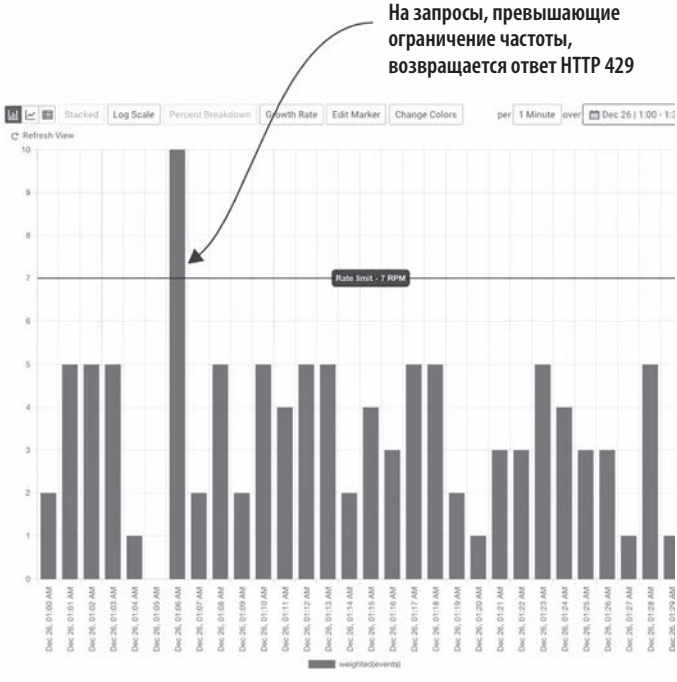


Рис. 11.27. Диаграмма временного ряда с количеством запросов за минуту и маркером ограничения частоты

Долгосрочные квоты API измеряют объем использования API потребителем за более длительный промежуток времени (например, часы, дни или месяцы) для проверки соответствия условиям контракта и целям монетизации. Например, клиенту может быть выделена квота в 10 000 запросов в месяц исходя из пакета продукта API, на который он подписался. Долгосрочные квоты API удобны для выставления счетов на основании фактического использования. Команда поддержки пользователей может отправить пользователю сообщение, чтобы

https://liveinternet.club

оповестить об исчерпании квоты и необходимости приобрести более объемный пакет API. Квоты API также полезны для блокирования намеренного злоупотребления автоматизированными процессами, использующими избыточные запросы API — например, парсеры или спам.

11.2.3. Инструменты мониторинга

Итак, вы узнали о технических метриках API, которые можно измерять в процессе мониторинга. Теперь я немного расскажу об инструментах.

В зависимости от способа развертывания инструменты мониторинга делятся на две категории — на базе агентов и безагентные. *Инструменты мониторинга на базе агентов* требуют установки в целевой системе облегченного агента для чтения данных мониторинга, передачи данных инструменту мониторинга и внедрения команд из инструмента мониторинга в отслеживаемую систему. Moesif — пример инструмента мониторинга на базе агентов. *Безагентные инструменты мониторинга* читают данные мониторинга, используя такие сетевые протоколы, как SMTP, SSH, специфические API приложений или встроенные агенты мониторинга для отслеживания работоспособности целевой системы. Иначе говоря, они не требуют установки стороннего программного агента для целевой системы. Некоторые API-шлюзы (такие, как Apigee) включают встроенную функциональность мониторинга API, но их также можно интегрировать со сторонними инструментами мониторинга посредством установки агента.

Как я уже говорил, существует множество проприетарных инструментов мониторинга, в числе которых AWS CloudWatch, New Relic, Datadog и Moesif. Prometheus (https://prometheus.io) — популярный инструмент мониторинга событий и уведомлений с открытым исходным кодом, который собирает и сохраняет данные временных рядов. Он часто подключается к Grafana (https://grafana.com) — системе аналитики и визуализации данных с открытым исходным кодом, предназначенной для создания дашбордов.

11.2.4. Оповещения и синтетический мониторинг

Мониторинг API сообщает о любой необычной активности или необычных паттернах в работе API. Примеры таких событий — неожиданные выбросы трафика или повышение задержки API. В процессе мониторинга API следует создавать оповещения, которые будут выдаваться при возникновении определенных событий. Оповещения должны базироваться на заранее заданных правилах, таких как появление множественных ошибочных ответов 5xx от API. Оповещения можно настроить для разных каналов: CMC, Slack, PagerDuty и т. д.

Еще одна важная разновидность — синтетический мониторинг, направленный на проактивное отслеживание доступности, эффективности и функциональности API посредством моделирования поведения пользователя в управляемой среде. Можно организовать моделирование путем выполнения скриптов или

https://liveinternet.club

автоматизированных тестов, которые имитируют типичные пользовательские взаимодействия с API. Например, такие тесты могут покрывать аутентификацию в API и выполнение типичных запросов API. Синтетический мониторинг должен выполняться регулярно, в идеале из разных мест (например, с использованием облачных дата-центров из различных географических регионов) для тестирования производительности и доступности API. Синтетический мониторинг позволяет организации проактивно выявлять проблемы взаимодействия с пользователями до того, как они затронут реальных пользователей.

Многие платформы наблюдаемости (например, New Relic, AppDynamics и Datadog) содержат функциональность синтетического мониторинга, но существуют и специализированные сервисы — такие, как Checkly (www.checklyhq.com).

11.2.5. Индикаторы уровня обслуживания, цели уровня обслуживания и соглашения об уровне обслуживания

Прежде чем завершить наше обсуждение мониторинга API, я хочу упомянуть некоторые важные концепции метрик обслуживания, которые часто используются специалистами SRE при обсуждении эффективности сервисов. Это SLI — индикатор уровня обслуживания (Service Layer Indicator), SLO — цель уровня обслуживания (Service Level Objective) и SLA — соглашение об уровне обслуживания (Service Level Agreement). SLI представляет собой количественную метрику некоторого параметра сервиса, используемого для измерения его работоспособности. Это, например, такие метрики, как задержка, частота ошибок сервера и пропускная способность. SLI играет важную роль, поскольку обеспечивает согласованную оценку эффективности работы сервиса.

SLI часто используются в сочетании со SLO и SLA. SLO — целевое значение или критерий, назначаемые провайдером сервиса для SLI. Это пороговое значение, которое сервис не должен превышать. Примером SLO задержки может быть обеспечение задержки 500 мс либо менее 99 % запросов к эндпоинту за 7-дневный период времени. Что касается SLA, это формальные контрактные соглашения с клиентами о достижении целевых значений SLI.

> **СОВЕТ** Еще одна концепция, связанная с SLO, — *бюджет ошибок*. Его можно представить как количество ошибок сервера, которые API может накопить за заданный период времени до того, как пользователи начнут ощущать дискомфорт. Обсуждение этой темы выходит за рамки книги, но вы можете обратиться к публикации «Establishing SRE Foundations: A Step-by-Step Guide to Introducing Site Reliability Engineering in Software Delivery Organizations» Владислава Укиса (Vladyslav Ukis) (Addison-Wesley Professional, 2022). Также узнать больше о SLA для API можно в статье «Pattern: Service Level Agreement» по адресу https://mng.bz/x2e8.

Значения SLI и SLO для API необходимо устанавливать с точки зрения клиента и настраивать мониторинг и оповещения на их основе. Но SLI и SLO должны эволюционировать по мере развития продукта API и опыта взаимодействия с ним.

https://liveinternet.club

11.3. Заключение

В этой книге вы узнали, что суть автоматизации разработки продукта API заключается в сквозной автоматизации рабочего потока создания API с использованием принципов GitOps и DevOps. Мы рассмотрели возможности автоматизации на разных стадиях рабочего потока, от проектирования до разработки, развертывания и тестирования, публикации и эксплуатации. Мы также рассмотрели такие концептуальные инструменты, как A3, предоставляющие схему применения улучшений, оценки эффекта изменений и извлечения полезных знаний.

> **СОВЕТ** Область автоматизации рабочего потока создания API и автоматизированного управления постоянно развивается. Если вы хотите оставаться в курсе последних методов, инструментов и отраслевых тенденций в этой области, подпишитесь на мою новостную рассылку: https://ikenna.co.uk/newsletter.

Пользуясь рассмотренными методами, вы сможете постоянно совершенствовать свой рабочий поток создания API для сокращения времени исполнения и улучшения метрик качества. По совету из главы 2 начните с планирования рабочего потока, а затем ищите возможности для его улучшения. Далее используйте методы, описанные в книге, для применения автоматизации.

Итоги

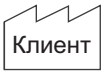
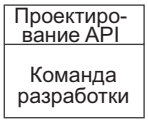
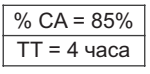
- Основой для принятия решений и определения эталонных целей улучшения продукта API служат метрики этого продукта.
- Карта пути разработчика наглядно представляет процесс, который проходит разработчик при знакомстве с вашим API и его использовании. С ее помощью можно определить важные метрики продукта, которые необходимо отслеживать на каждой стадии, чтобы представлять общую картину опыта разработчика.
- Инструменты аналитики продукта API помогут сохранять, визуализировать и распространять важные метрики продукта в команде. К числу таких метрик относятся количество регистраций новых пользователей, TTFHW, DAU, TTFT, удержание пользователей, коэффициент конверсии и ключевые потребители API.
- Метод RED — популярная формула мониторинга, которую можно применять для мониторинга технической производительности API. Центральное место в нем занимают метрики частоты запросов, частоты ошибок и продолжительности обработки.
- Ограничения частоты определяют точку насыщения API; установите их, чтобы защитить свой API от чрезмерной нагрузки, которая может ухудшить взаимодействие с другими пользователями.
- Чтобы организовать мониторинг и рассылку оповещений, необходимо определить значение SLO для API.

https://liveinternet.club

Приложение А
Значки карты потока создания ценности

В этом приложении перечислены значки картирования потока создания ценности (VSM, Value Stream Mapping), которые вы можете использовать в своих картах процессов. При желании добавляйте к ним любые другие значки, которые вам понадобятся для представления потоков работы и информации в вашей системе, но не забудьте определить их значение. Создание карты процесса — искусство, а не наука, и важно с ее помощью обеспечить общее понимание потока работы и данных. Помните, что карта должна быть простой и толковаться однозначно.

Таблица А.1. Значки карты потока создания ценности

Значок	Представляет	Описание
	Источники за пределами процесса	Внешние и внутренние клиенты и поставщики процесса
	Шаг процесса	Шаг процесса, на котором выполняется задача. В верхней части приводится его описание (название), а в нижней — название команды или роли, выполняющей задачу
	Блок данных	Используется для представления важных данных о процессе
	Информационная система	IT-системы, обеспечивающие прохождение информации по потоку ценности
	Ручной поток информации	Поток информации между информационными системами и людьми. Стрелка показывает направление потока

https://liveinternet.club

Значок	Представляет	Описание
	Автоматизированный поток информации	Автоматизированный поток информации между информационными системами. Стрелка показывает направление потока
	Незавершенная работа	Частично завершенная работа, находящаяся в очереди между шагами процесса в ожидании обработки. Обычно представлена рабочими тикетами в журнале незавершенной работы. Количество элементов записывается под значком
	Push-система	Заштрихованная стрелка обозначает перемещение работы к следующему шагу процесса без проверки того, способен ли следующий шаг принять ее
	Извлечение	Круговая стрелка обозначает шаг, получающий работу от предыдущего процесса (или из супермаркета)
	Планирование по принципу Go-see («иди и смотри»)	Корректировка приоритетов работы на основе ручного мониторинга уровня незавершенной работы
	Релиз	Выпуск готового продукта на продакшен для клиентов
	Супермаркет	Управляемый склад компонентов, которые могут загружаться последующим процессом для выполнения необходимой работы

https://liveinternet.club

Б Установка инструментов линтинга API и выявления различий OpenAPI

В этом приложении описан процесс установки инструментов, необходимых для выполнения примеров из книги. Репозиторий кода книги доступен по адресу https://github.com/apiopsbook/apiops.

Чтобы установить необходимые инструменты на своем компьютере, выполните инструкции, приведенные ниже. На случай если вы знакомы с технологией Docker, я также подготовил образ Docker со всеми установленными инструментами командной строки. Чтобы установить инструменты, не использующие интерфейс командной строки (CLI), — такие, как Visual Studio Code (VS Code), GitHub Copilot и Anthropic Claude, — придется выполнить инструкции. Создайте контейнер Docker и присоедините его к терминалу, выполнив следующие скрипты:

```
git clone https://github.com/apiopsbook/apiops
cd apiops
./2_create_apiops_container.sh
./3_attach_to_apiops_container.sh
```

В результате вы подключитесь к терминалу контейнера и сможете обратиться к файлам с исходным кодом книги на своем хосте, перейдя в папку `books-source`:

```
root@apiops-host:/#
root@apiops-host:/# cd book-source/
root@apiops-host:/book-source# ls -l
```

Отключение от терминала контейнера осуществляется командой `exit`. Но если вы хотите установить инструменты командной строки на своем компьютере, выполните инструкции, приведенные ниже.

https://liveinternet.club

Б.1. Установка Node.js

Node.js — исполнительная среда JavaScript с открытым исходным кодом, выполняемая на базе ядра V8 JavaScript вне браузера. Node Package Manager (NPM) — стандартный менеджер пакетов для Node.js, он включен в установку Node.js. Чтобы установить Node.js, загрузите и запустите установочный пакет для своей операционной системы по адресу https://nodejs.dev/en/download. Чтобы проверить установку Node.js, выполните следующие команды в своем терминале:

```
node -v
npm -v
```

Б.2. Установка Spectral

Spectral (https://github.com/stoplightio/spectral) — линтер JSON и YAML для файлов определений OpenAPI и AsyncAPI. Spectral можно установить командой `npm`:

```
$ npm install -g @stoplight/spectral-cli
```

Информацию о других методах установки Spectral с использованием Yarn, Docker или платформенных двоичных файлов см. на сайте Stoplight (https://mng.bz/v8yM).

Чтобы проверить установку Spectral, в каталоге `appendix` запустите линтинг Spectral на минимальном файле определений API с использованием базового набора правил Spectral OpenAPI. Минимальный файл определения API, предоставленный для этой цели, действителен, но он не соответствует трем рекомендациям правил из встроенного в Spectral набора правил спецификации OpenAPI; соответственно, он выдает предупреждения, демонстрирующие, как Spectral сообщает о проблемах:

```
spectral lint minimal.oas.yaml
```

Вывод должен выглядеть так:

```
1:1  warning  oas3-api-servers  OpenAPI "servers" must be present
➥ and non-empty array[1].
 2:6  warning  info-contact      Info object must have "contact"
➥ object[2].    info
 2:6  warning  info-description  Info "description" must be present
➥ and non-empty string[3].  info
✖ 3 problems (0 errors, 3 warnings, 0 infos, 0 hints)[4]
```

---

[1] OpenAPI “servers” must be present and non-empty array — Значение servers OpenAPI должно присутствовать и быть непустым. — *Примеч. пер.*

[2] Info object must have “contact” object — Объект info должен содержать объект contact. — *Примеч. пер.*

[3] Info “description” must be present and non-empty string — Описание (description) должно присутствовать и быть непустой строкой. — *Примеч. пер.*

[4] 3 problems (0 errors, 3 warnings, 0 infos, 0 hints) — 3 проблемы (0 ошибок, 3 предупреждения, 0 информации, 0 подсказок). — *Примеч. пер.*

https://liveinternet.club

Б.3. Установка Visual Studio Code

VS Code — популярный бесплатный редактор от Microsoft. Его можно загрузить и установить по адресу https://code.visualstudio.com/download. VS Code имеет интерфейс командной строки CLI, который запускается командой code и позволяет открывать файлы, устанавливать расширения и выводить диагностическую информацию из командной строки. В установках VS Code для Windows и Linux код командной строки включен по умолчанию. Возможно, для macOS вам придется добавить VS Code в путь вручную. Инструкции, как это сделать, см. на странице https://mng.bz/ZEQm.

Б.4. Установка плагинов Spectral Visual Studio Code

Расширение Spectral VS Code от Stoplight позволяет запускать линтинг Spectral в редакторе в процессе редактирования определений API. Вы можете указать, хотите ли вы получать оповещения о проблемах линтинга во время ввода или же при сохранении файла. Чтобы установить расширение через интерфейс VS Code, выполните следующую команду:

```
code --install-extension stoplight.spectral
```

Также можно установить расширение через Visual Studio Marketplace, кликнув на кнопке Install (https://mng.bz/RZ1j).

Б.5. Установка 42Crunch OpenAPI (Swagger) Editor

Плагин VS Code OpenAPI (Swagger) Editor от 42Crunch позволяет редактировать файлы определений OpenAPI с поддержкой IntelliSense, линтинга, проверки соответствия схеме, навигации по коду и предварительного просмотра. 42Crunch (https://42crunch.com) — коммерческая платформа безопасности API. Помимо редактирования файлов определений API, этот плагин предоставляет важную функциональность, позволяющую запускать API Security Audit на платформе 42Crunch.

Плагин можно установить через интерфейс VS Code, выполнив следующую команду:

```
code --install-extension 42Crunch.vscode-openapi
```

Также его можно установить через Visual Studio Marketplace, кликнув на кнопке Install (https://mng.bz/2Km8), или прямым поиском OpenAPI (Swagger) Editor в представлении Extensions на панели VS Code Activity. Ссылка на маркетплейсе предоставляет рекомендации по использованию плагина для создания новых файлов OpenAPI, навигации по определениям OpenAPI, предварительному просмотру определений OpenAPI и использованию IntelliSense.

https://liveinternet.club

Плагин также позволяет рендерить файл OpenAPI. Кликните на кнопке OpenAPI: Show Preview Using the Default Renderer, чтобы увидеть предварительное представление определения API в формате Swagger UI на правой панели.

Я рекомендую настроить плагин для выполнения с файлами операции Security Audit. Для этого кликните на фиолетовой кнопке 42c в правом верхнем углу. При первом клике вам будет предложено ввести свой адрес электронной почты, чтобы компания 42Crunch прислала вам токен доступа (рис. Б.1). После получения вставьте токен в соответствующую подсказку в VS Code.

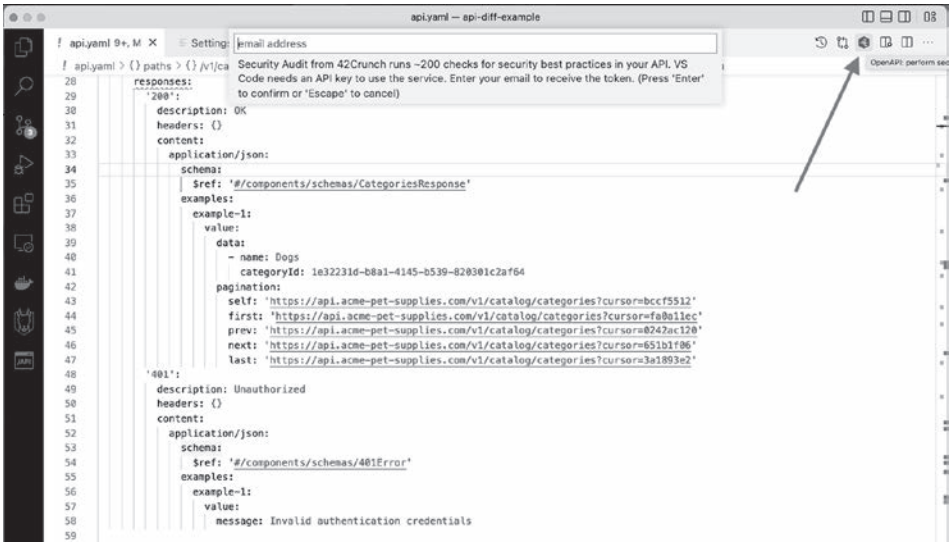


Рис. Б.1. Запуск Security Audit кнопкой 42c

Теперь кликните на фиолетовой кнопке OpenAPI: Perform Security Audit в правом верхнем углу. Кнопка сканирует ваше определение на наличие уязвимостей и выведет отчет, как на рис. Б.2.

Б.6. Установка jq и yq

jq (https://jqlang.github.io/jq) — удобный JSON-процессор для фильтрации и преобразования разметки JSON. Он получает на входе поток данных JSON, применяет фильтрацию и логику преобразования и генерирует результат. Чтобы установить jq, можно воспользоваться менеджером пакетов, подходящим для вашей операционной системы. Пользователи Mac могут воспользоваться командой `brew install jq`. В Windows установите его при помощи менеджера пакетов Chocolatey, выполнив команду `choco install jq`. В Ubuntu Linux выполните команду `sudo apt install -y jq`.

https://liveinternet.club

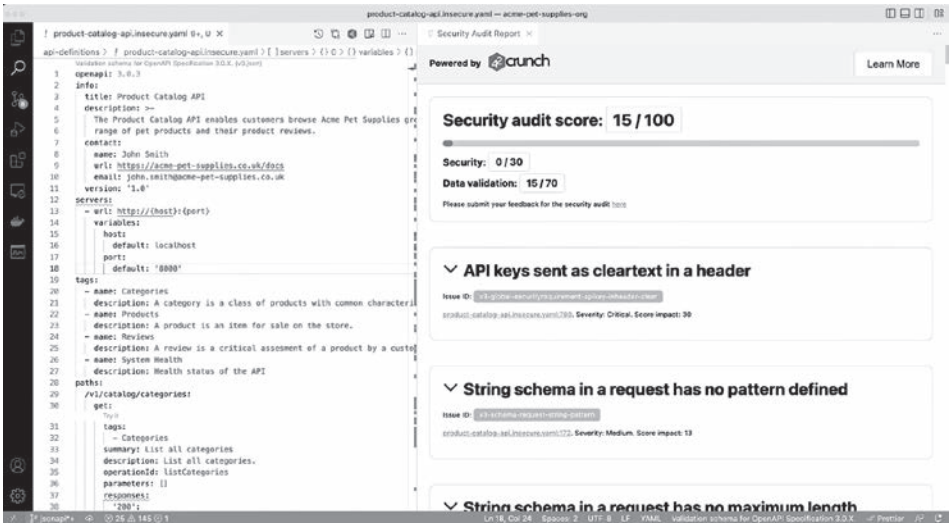


Рис. Б.2. Открытый аудит безопасности файла определения API оценивает уровень безопасности и уведомляет о возможных уязвимостях в файле определения API

yq (https://github.com/mikefarah/yq) — процессор YAML, JSON и XML, который также может использоваться для фильтрации и преобразования структурированных данных. Кроме того, он подходит для преобразования YAML в JSON (и наоборот). Его синтаксис близок к синтаксису jq. yq написан на языке Go.

Для установки yq в macOS можно воспользоваться командой `brew install yq`; в Windows используется команда `choco install yq`, а в Linux — команда `snap install yq`. Информацию о других способах установки см. на странице https://github.com/mikefarah/yq/#install.

Чтобы протестировать установку yq, попробуйте преобразовать файл `test.yaml` в JSON командой `yq -Poj test.yaml`. Чтобы преобразовать файл `test.json` в YAML, выполните команду `yq -Poy test.json`.

Б.7. Установка jpp

jsonpath-cli (www.jsware.io/jsonpath-cli/) — интерфейс командной строки для JSONPath Plus. Он запускается командой `jpp` и может использоваться для преобразования и избирательного извлечения данных из документов JSON. Его можно установить при помощи `npm`:

```
npm install -g @jsware/jsonpath-cli
```

Проверьте установку, выполнив команду с флагом `-h` для вывода опций использования:

```
jpp -h
```

https://liveinternet.club

Б.8. Установка Tufin/oasdiff

Tufin/oasdiff (https://github.com/Tufin/oasdiff) — инструмент для поиска различий OpenAPI 3.x, работающий в режиме командной строки. В примерах этой книги используется Tufin/oasdiff версии 1.6.2. Если вы работаете в macOS, установите Tufin/oasdiff при помощи Homebrew:

```
brew tap tufin/homebrew-tufin
brew install oasdiff
```

Чтобы команды сработали, возможно, придется сначала установить пакет Xcode Command Line Tools, если это не было сделано ранее. Для установки можно воспользоваться командой `xcode-select –install`.

Код Tufin/oasdiff написан на Go. Если вы уже используете Go на своем компьютере, Tufin/oasdiff также можно установить такой командой:

```
go install github.com/tufin/oasdiff@latest
```

Кроме того, доступны двоичные файлы для Windows, Linux и macOS. Информацию о том, как их загрузить, см. на странице https://github.com/Tufin/oasdiff.

Установку можно протестировать следующей командой:

```
oasdiff -help
```

Б.9. Установка swagger-ui-cli

Swagger-ui-cli (https://github.com/egomobile/swagger-ui-cli) — удобный инструмент для рендеринга файлов определений OpenAPI из командной строки. Его можно установить следующей командой:

```
npm install -g swagger-ui-cli
```

Протестируйте его на файле определения API, выполнив команду:

```
swagger-ui products-api.json
```

Она должна открыть браузер на странице http://127.0.0.1:8080 и сгенерировать файл определения API.

Б.10. GitHub Copilot

GitHub Copilot — ИИ-ассистент для кодинга, разработанный GitHub в сотрудничестве с OpenAI. Он генерирует фрагменты кода и предложения для автозавершения на различных языках программирования.

https://liveinternet.club

Для использования Copilot с VS Code вам понадобится активная подписка Copilot — либо персональная, либо в рамках корпоративной подписки GitHub Copilot Business. Прежде чем оформлять платную подписку, можно воспользоваться одноразовым 30-дневным пробным периодом. Подробности о том, как зарегистрироваться для бесплатного пробного периода и как настроить GitHub Copilot в VS Code через расширение Copilot, см. на странице https://mng.bz/1Gp1.

Также установите расширение GitHub Copilot Chat для VS Code. GitHub Copilot Chat — вспомогательный чат-бот для GitHub Copilot. В Copilot Chat можно задавать вопросы и получать ответы на вопросы по таким темам, как синтаксис языков программирования, концепции и т. д. Он может использоваться для генерации юнит-тестов, объяснения кода или предложений по исправлению и улучшению кода. Инструкции по установке расширения GitHub Copilot Chat для VS Code см. на странице https://mng.bz/PZrY.

Б.11. Регистрация в Anthropic Claude

Claude — ИИ-чат-бот от Anthropic. Как и другие чат-боты (например, OpenAI ChatGPT или Googles Bard), Claude имеет интерфейс обмена сообщениями, в котором можно задавать вопросы и получать подробные, актуальные ответы. Разработчики Claude постарались сделать его безопасным, полезным и честным. Для этого он использует обратную связь от ИИ для оценки своих результатов. Обратная связь от ИИ базируется на наборе принципов, относящихся к определенным темам и называемыми «конституциями», поэтому Anthropic называет этот механизм «конституционным ИИ».

Чтобы зарегистрироваться, перейдите по адресу https://claude.ai, выполните инструкции по регистрации и используйте интерфейс для общения с Claude в браузере. На момент написания книги Claude не был доступен в некоторых частях Европы. Список регионов, в которых доступен Claude, находится по адресу www.anthropic.com/claude-ai-locations.

https://liveinternet.club

Б Краткое введение в JSON Pointer

В этом приложении приводится краткое описание JSON Pointer. Чтобы поработать с приведенными примерами, воспользуйтесь репозиторием проекта, клонированным в приложении Б. Перейдите в папку `appendix` в проекте и найдите файл `products-api.json`, который вам для этого понадобится.

Spectral использует синтаксис JSON Pointer для определения части документа JSON, к которой должно применяться переопределение. Этот синтаксис предназначен для поиска значения в документе JSON, а спецификация определяется в RFC6901. Он представляет собой строку, содержащую ноль или более токенов с префиксом \. Чтобы увидеть его в действии, вычислите выражения JSON Pointer в документе JSON. Установите `json-joy` (https://github.com/streamich/json-joy), чтобы получить доступ к инструменту командной строки `json-pointer` для тестирования выражений JSON Pointer:

```
npm install -g json-joy
```

У вас есть следующий документ `products-api.json` в папке `appendix`:

```
{
  "openapi": "3.0.3",
  "info": {
    "title": "An example API[1]",
    "version": "0.0.1"
  },
  "paths": {
    "/v1/products/{id}": {
      "delete": {
        "parameters": [
          {
            "in": "path",
```

[1] An example API — Пример API. — *Примеч. пер.*

https://liveinternet.club

```
            "name": "id",
            "required": true,
            "schema": {
              "type": "integer"
            }
          }
        ],
        "responses": {
          "204": {
            "description": "Product deleted[1]."
          }
        }
      }
    }
  }
}
```

Чтобы вычислить указатель `/openapi`, выполните следующую команду:

```
cat products-api.json | json-pointer /openapi
```

Вы получите результат:

```
"3.0.3"
```

Чтобы вычислить указатель `/info`, выполните команду:

```
cat products-api.json | json-pointer /info
```

Результат будет таким:

```
{
    "title": "An example API",
    "version": "0.0.1"
}
```

Для указателя `/info/title` вы получите следующий результат:

```
"An example API"
```

Символы `~` и `/` имеют особый смысл в синтаксисе JSON Pointer; при вхождении в токены они должны дополнительно кодироваться. `~` кодируется в виде `~0`, а `/` — в виде `~1`. Обращайте на это внимание при определении выражения JSON Pointer для объекта внутри объекта пути OpenAPI — токен элемента пути должен быть закодирован. Чтобы увидеть, как это работает, выполните следующую команду в своей командной строке:

```
cat products-api.json |
?json-pointer /paths/~1v1~1products~1{id}/delete/responses
```

[1] Product deleted — Продукт удален. — *Примеч. пер.*

https://liveinternet.club

Вы получите такой результат:

```
{
    "204": {
        "description": "All products deleted[1]."
    }
}
```

Кроме того, можно обращаться к элементам массива с указанием их индекса. При выполнении следующей команды вы получите на выходе `"path"`:

```
cat products-api.json |
?json-pointer /paths/~1v1~1products~1{id}/delete/parameters/0/in
```

Выражение JSON Pointer может быть строковым значением JSON, как показано здесь, или идентификатором фрагмента URI (RFC3986). Например, если документ JSON хранится в файле с именем `myapi.json`, приведенное ниже выражение является действительной ссылкой на указатель для выборки хранящегося в нем элемента:

```
myapi.json#/paths/~1v1~1products~1{id}/ /delete/responses/204/description.
```

Данная возможность используется в Spectral. Однако вы не сможете протестировать ее с использованием интерфейса командной строки `json-joy json-pointer`, так как он ожидает получить значение JSON из стандартного ввода (STDIN):

> **ПРИМЕЧАНИЕ** Кроме командной строки, выражения JSON Pointer можно вычислять на сайте www.jsonquerytool.com.

[1] All products deleted — Все продукты удалены. — *Примеч. пер.*

https://liveinternet.club

Г Инструменты проверки соответствия API и аналитики

В этом приложении рассматривается еще ряд инструментов. Если вы используете контейнер Docker, о котором я упоминал в приложении Б, обратите внимание на то, что названные здесь инструменты (кроме Moesif, требующего регистрации) уже установлены в нем. Но если вы захотите установить инструменты на своем локальном компьютере, выполните инструкции из этого приложения.

Г.1. Установка и запуск Kong

Kong (https://github.com/Kong/kong) — популярный API-шлюз с открытым исходным кодом, известный своей высокой производительностью. Kong использует два файла конфигурации: `kong.yaml` и `kong.conf`. Файл `kong.yaml` предназначен для настройки сущностей Kong. Это информация о конфигурации, описывающая, как Kong маршрутизирует запросы, например к вышестоящим сервисам, маршрутам, плагинам и потребителям API. Файл `kong.conf` предназначен для настройки свойств сервера Kong, таких как префикс рабочего каталога, расположение файла журнала, кластер и параметры базы данных. Kong работает в двух режимах: с базой данных или без нее. В книге Kong используется в режиме без базы данных.

Если вы работаете в macOS или Linux, Kong можно установить напрямую на вашем компьютере. Kong не поддерживает прямую установку в Windows. Следовательно, пользователям Windows придется выполнять Kong через Docker. Ниже приведены инструкции выполнения Kong в macOS и Docker. Информацию о других вариантах установки см. на странице https://docs.konghq.com/gateway/latest/install.

Г.1.1. Установка Kong для macOS

Установите Kong при помощи менеджера пакетов Homebrew, выполнив следующие команды:

```
brew tap kong/kong
brew install kong
```

https://liveinternet.club

В каталоге `appendix` нашего проекта протестируйте установку Kong командой:

```
kong start --conf kong.conf
```

Файл `kong.conf` в каталоге `appendix` настроен для создания рабочего каталога с именем `kong_dir`, в котором Kong размещает временные файлы и журналы. По умолчанию Kong прослушивает общедоступный трафик на порте 8000, а трафик административных запросов — на своем внутреннем административном REST API на порте 8001. Протестируйте установку следующим запросом, маршрутизируемым к вышестоящему сервису http://httpbin.org/get:

```
curl http://localhost:8000/mock
```

Ответ должен выглядеть примерно так:

```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.79.1",
    "X-Amzn-Trace-Id": "Root=1-63ba204c-33af0b84781054ba6d144785",
    "X-Forwarded-Host": "localhost",
    "X-Forwarded-Path": "/mock",
    "X-Forwarded-Prefix": "/mock"
  },
  "origin": "127.0.0.1, 2.124.41.27",
  "url": "http://localhost/get"
}
```

Остановите Kong следующей командой:

```
kong stop -p kong_dir
```

Г.1.2. Запуск Kong через Docker

Материал этого раздела предполагает, что Docker уже работает на вашем компьютере. Docker — программная платформа, использующая виртуализацию уровня ОС для разработки, поставки и запуска программ в пакетах, называемых контейнерами. О том, как установить приложение Docker Desktop для своей операционной системы, см. на странице https://docs.docker.com/get-docker.

Создайте и запустите контейнер Kong Docker с именем `kong-dbless` при помощи скрипта `run_docker_container.sh`:

```
./run_kong_docker.sh
```

На Windows вместо этого следует запустить скрипт `run_kong_docker.bat`:

```
C:\Users\dev\appendix>run_kong_docker.bat
```

Откройте в браузере страницу http://localhost:8000/mock; вы увидите такой же ответ, как в приведенном выше примере для macOS.

https://liveinternet.club

Чтобы остановить выполняемый контейнер Kong, выполните следующую команду:

```
docker stop kong-dbless
```

Чтобы перезапустить контейнер, снова выполнять скрипт `run_docker_container.sh` не нужно. Просто выполните команду `docker start` для контейнера `kong-dbless`:

```
docker start kong-dbless
```

Г.2. Установка Java Development Kit и Maven

Для запуска кода Java, приведенного в книге, и инструмента OpenAPI Generator вам понадобится пакет JDK (Java Development Kit) версии 20 или выше. Чтобы установить JDK 20, перейдите на страницу Oracle JDK https://docs.oracle.com/en/java/javase/20/index.html и кликните на ссылке `Installation Guide`. Выполните инструкции по установке JDK для своей операционной системы.

Maven — система автоматизации сборки проектов Java. Чтобы установить Maven, загрузите дистрибутив Maven по адресу https://maven.apache.org/install.html и выполните инструкции на этой странице, чтобы разархивировать пакет и добавить каталог `bin` в переменную среды `PATH`. Протестируйте установку Maven, выполнив следующую команду для проверки версии:

```
mvn --version
```

Г.3. Установка OpenAPI Generator CLI

OpenAPI Generator CLI (https://github.com/OpenAPITools/openapi-generator-cli) — приложение командной строки, которое загружает и запускает инструмент OpenAPI Generator. Обратите внимание: OpenAPI Generator зависит от Java, поэтому перед установкой и запуском OpenAPI Generator CLI в вашей системе должна быть установлена поддержка Java (как описано в предыдущем разделе). Кроме того, OpenAPI Generator CLI требует, чтобы исполняемый файл Java находился в каталоге из переменной среды `PATH`.

Чтобы установить OpenAPI Generator CLI, выполните следующую команду:

```
npm install -g @openapitools/openapi-generator-cli
```

Протестируйте установку командой, которая выводит версию CLI:

```
openapi-generator-cli version
```

Г.4. Установка Redocly CLI

Redocly CLI (https://github.com/Redocly/redocly-cli) — инструмент для управления определениями OpenAPI. Он содержит мощный линтер, средства фильтрации, разбиения, группировки и вычисления статистики. Он может генерировать справочную документацию API в динамическом формате на основе определений

https://liveinternet.club

OpenAPI в формате Redocly (https://github.com/Redocly/redoc). В этой книге используется Redocly CLI 1.8.2. Установите CLI, выполнив команду:

```
npm install -g @redocly/cli
```

Также установите сервер HTTP Node для предоставления файлов HTML следующей командой:

```
npm install -g http-server
```

Г.5. Установка Schemathesis

Schemathesis (https://schemathesis.readthedocs.io/en/stable) — инструмент тестирования на базе спецификаций и свойств для OpenAPI и GraphQL API. Чтобы установить его, сначала необходимо установить на компьютере поддержку Python; это можно сделать на странице www.python.org/downloads. Затем для установки Schemathesis выполните команду:

```
pip install schemathesis
```

В этой книге используется Schemathesis 3.25.0. Если у вас установлена более старая версия Schemathesis, ее можно обновить следующей командой:

```
pip install schemathesis --upgrade
```

Г.6. Установка Portman

Portman (https://github.com/apideck-libraries/portman) преобразует спецификации OpenAPI в коллекции Postman. В этой книге используется Portman 1.22.0. Чтобы установить его, выполните следующую команду:

```
npm install -g @apideck/portman
```

Г.7. Установка Prism

Prism — мок-сервер и валидирующий прокси-сервер для определений OpenAPI и коллекций Postman. На момент написания книги для запуска Prism требовалась версия NodeJS 16 и выше (для NodeJS 18.x требуется версия 18.16 и выше). В этой книге используется Prism версии 5.5.4.

Чтобы установить Prism, выполните следующую команду:

```
npm install -g @stoplight/prism-cli
```

Г.8. Настройка учетной записи Moesif

Moesif — инструмент аналитики продуктов API, который используется в примерах аналитики и мониторинга API из главы 11. Выполните установку, описанную ниже, когда перейдете к этой главе.

https://liveinternet.club

Чтобы создать учетную запись Moesif, откройте сайт www.moesif.com и создайте бесплатную учетную запись. В модальном окне для ввода имени приложения введите `ProductCatalogAPI`. На открывшейся странице Quick Install выводится список установок для серверных интеграций. Не обращайте на него внимания и кликните на кнопке Demo в окне No Events Collected Yet.

В открывшемся представлении дашборда переключитесь с приложения Moesif Demos на свое приложение ProductCatalogAPI. Затем перейдите к настройкам своей учетной записи и скопируйте свой идентификатор Moesif Collector API ID, как показано на рис. Г.1. Этот ключ API необходим для работы моделирующих скриптов рабочего потока.

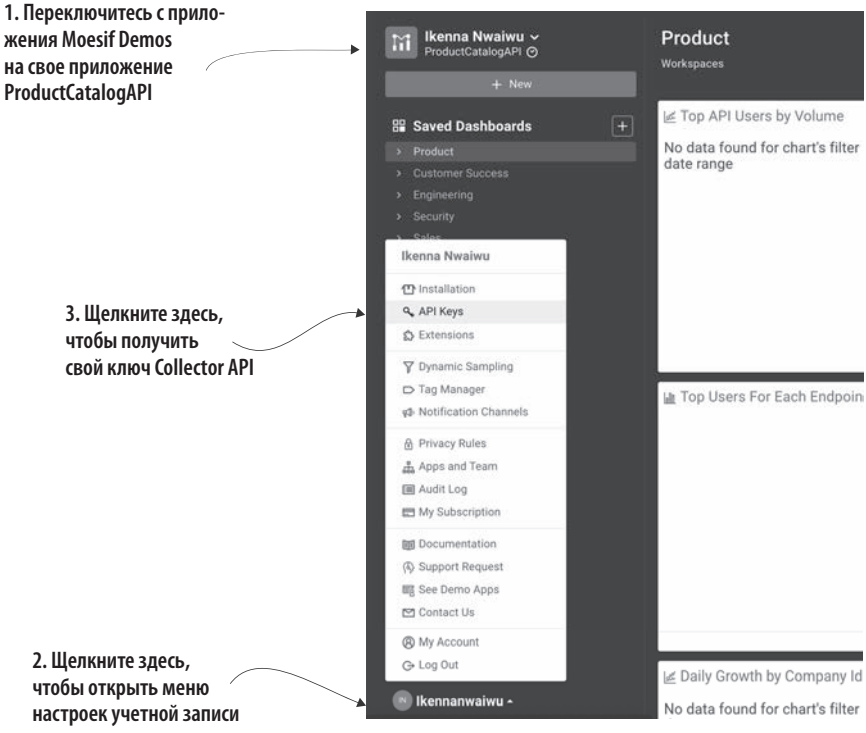


Рис. Г.1. Щелкните в меню профиля, чтобы получить ключи Moesif API

Moesif использует ключи API двух видов: идентификатор Collector Application ID для сбора данных и ключ Management API для обращения к данным в Moesif. Для запуска примеров из главы 11 необходимо добавить Moesif Collector Application ID как секрет в репозиторий GitHub. Существуют две разновидности Collector Application ID — для доверенных приложений и для других приложений (например, браузеров). Получите идентификатор для доверенных приложений и выполните инструкции со страницы документации GitHub (https://mng.bz/pplz), чтобы добавить его как секрет репозитория с именем `MOESIF_API_KEY`.

https://liveinternet.club

Д Docker и Kubernetes

Kubernetes — ведущая, открыто распространяемая платформа оркестрации и управления кластерами. Некоторые примеры этой главы включают запуск контейнеров сервиса API и API-шлюза в Kubernetes с использованием `minikube`. Подробное рассмотрение Kubernetes выходит за рамки книги, но я коротко познакомлю вас с этой технологией в данном приложении. Если вы уже работаете с Kubernetes, это приложение можно пропустить.

Д.1. Знакомство с Docker

Docker — открытая платформа, использующая виртуализацию уровня ОС для доставки программного обеспечения в стандартизированных модулях, которые называются *контейнерами* и содержат все необходимое для работы программы. *Образ контейнера* представляет собой статический файл с кодом, системными инструментами, системными библиотеками и настройками, необходимыми для создания контейнера в вычислительной системе. Программа, которая выполняет образ контейнера и преобразует его в работающий контейнер, называется *исполнительной средой* контейнера. В Docker такой средой является Docker Engine. Вместо виртуализации нижележащего компьютерного оборудования, как это делают виртуальные машины, контейнеры виртуализируют ОС (процессор, память, дисковое пространство и сетевые ресурсы), создавая представление ОС, изолированное от других приложений, как показано на рис. Д.1. Контейнеры обладают полноценной портируемостью (в отличие от традиционных развертываний ПО); они помогают разработчикам быстро создавать, тестировать и развертывать приложения.

https://liveinternet.club

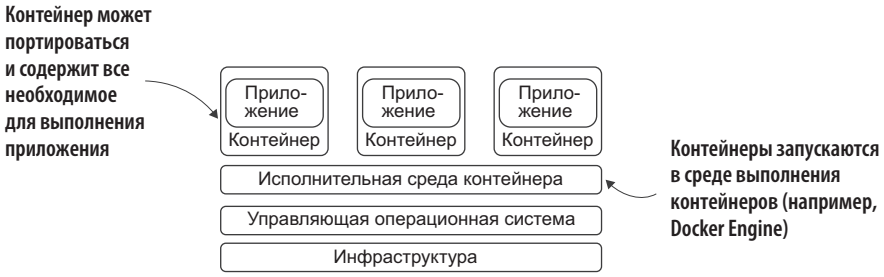


Рис. Д.1. Исполнительные среды контейнеров — такие, как Docker Engine, — обеспечивают виртуализацию на уровне ОС и выполняют контейнеры, имеющие доступ к изолированным системным ресурсам

Д.2. Установка Docker

Если в приложении Г вы выбрали вариант запуска Kong через Docker (например, вы работаете в Windows), то вы уже прошли процесс установки Docker на своем компьютере. Если вы еще не установили Docker, см. инструкции по установке Docker Desktop — приложения, содержащего набор инструментов Docker, — по адресу https://docs.docker.com/get-docker.

Д.3. Настройка сервисов в Docker Compose

Один из инструментов, включенных в поставку Docker Desktop, — Docker Compose. Docker Compose предназначен для управления сразу несколькими контейнерами Docker. С его помощью можно задать настройки разных контейнеров для сервисов вашего приложения в файле YAML, чтобы запустить их все одной командой. Вы можете запускать и останавливать сервисы, просматривать их статус, получать журнальный вывод выполняемых сервисов и выполнять ситуативные команды с ними. Docker Compose становится доступным после установки Docker Desktop.

Файл Docker Compose `chapter9/compose.yaml` содержит настройки разных сервисов, необходимых для вашего пайплайна развертывания. Он определяет сервисы, необходимые для запуска API-шлюза Kong и реестра API Apicurio. Вы будете запускать их из своего контейнера GitHub Actions. Но чтобы увидеть, как они работают локально, их можно запустить в фоновом режиме следующей командой:

```
docker compose --file chapter9/compose.yaml up --detach
```

Чтобы увидеть список запущенных сервисов, выполните команду:

```
docker compose --file chapter9/compose.yaml ls
```

https://liveinternet.club

Для остановки сервисов и удаления контейнеров используется команда:

```
docker compose --file chapter9/compose.yaml down
```

Д.4. Введение в Kubernetes

Kubernetes (также встречается сокращенное название K8s; https://kubernetes.io) — платформа с открытым исходным кодом для оркестрации распределения, планирования и выполнения приложений в компьютерном кластере. Kubernetes координирует кластер компьютеров, позволяя пользователям развертывать контейнеризованные приложения и управлять ими.

Кластер Kubernetes состоит из *управляющего слоя* (control plane) для управления кластером и рабочих машин, на которых выполняются контейнеризованные приложения. Эти рабочие машины называются *узлами*. У каждого узла имеется *kubelet* — агент, который управляет узлом и взаимодействует с управляющим слоем. У каждого узла также имеется исполнительная среда для выполнения контейнеров. Примером исполнительной среды контейнеров является Docker Engine. *Под* (pod) представляет собой группу контейнеров приложений и общих ресурсов (например, томов и IP-адресов) для этих контейнеров. На узле могут выполняться сразу несколько подов. Управляющий слой Kubernetes обеспечивает планирование выполнения подов по узлам кластера. Он предоставляет REST API для определения, развертывания и управления жизненным циклом контейнеров. Управляющий слой состоит из нескольких компонентов, включающих *сервер API*, *планировщик* и *менеджер контроллера*. Сервер API предоставляет REST API. Планировщик распределяет поды между узлами, а менеджер контроллера представляет собой процесс-демон, управляющий контроллерами Kubernetes. Инструмент командной строки Kubernetes kubectl используется для управления Kubernetes. Он взаимодействует с сервером API и может использоваться для выполнения команд для развертывания приложений, просмотра журналов и анализа ресурсов кластера. Архитектура Kubernetes представлена на рис. Д.2.

Под выполняется в приватной изолированной сети кластера Kubernetes. Под создается и обновляется при *развертывании*. Развертывание описывает, какие образы должны использоваться для контейнера приложения в поде, количество выполняемых подов в любой момент времени (то есть количество *реплик*) и способы обновления подов. Поды доступны только по своим внутренним IP-адресам в кластере Kubernetes. Для маршрутизации запросов к группе подов, выполняющих приложение, необходим *сервис*. Ресурс сервиса — именованная абстракция программного сервиса, состоящая из логического набора подов и политик для обращения к ним. Сервис имеет IP-адрес, внутренний для кластера.

https://liveinternet.club

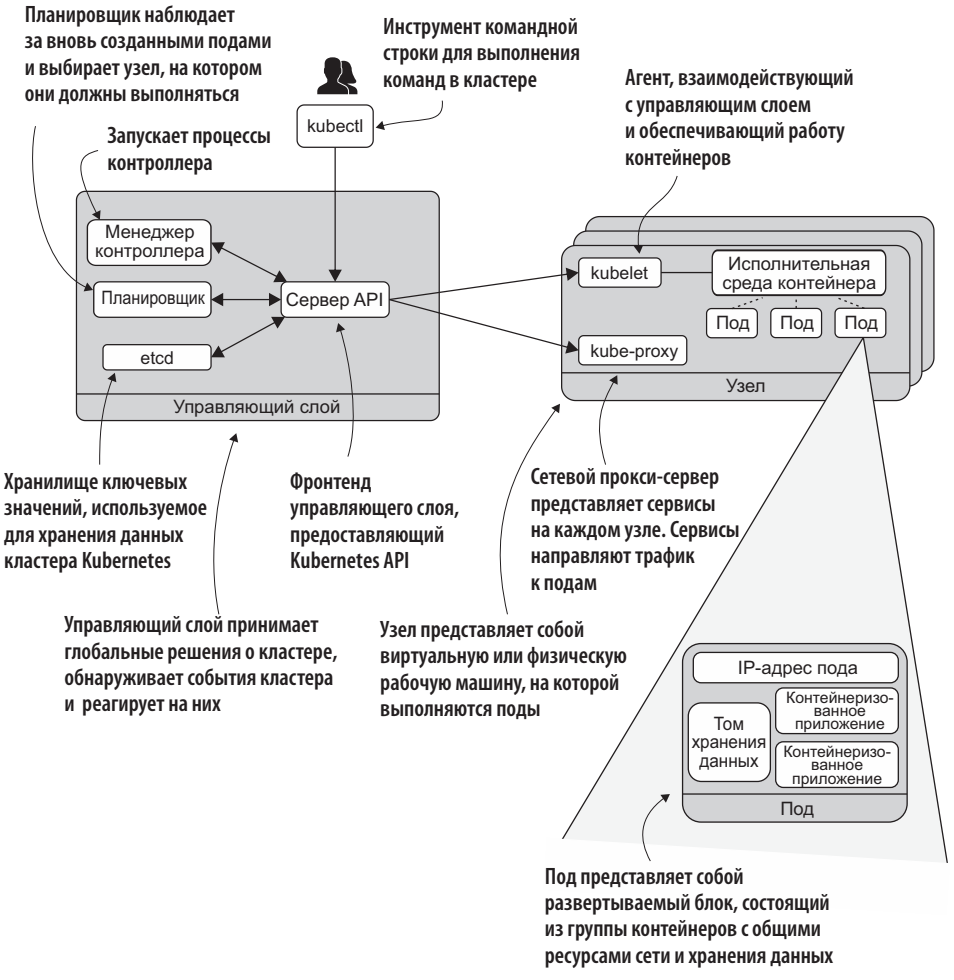


Рис. Д.2. Архитектура Kubernetes с управляющим слоем и рабочими узлами

Ingress-объект описывает набор правил, разрешающих входящие подключения HTTP/HTTPS к сервисам через URL-адреса с внешней доступностью. Ingress-объекты определяют *трафик «север/юг»*, то есть трафик из-за пределов кластера к объектам сервисов внутри кластера. Они обеспечивают завершение SSL, балансировку нагрузки трафика и маршрутизацию с учетом контента. *Ingress-контроллер* запускает процесс, реализующий ingress-объект. На рис. Д.3 показана маршрутизация внешнего трафика к подам с использованием ingress-объекта и сервиса. Примеры ingress-контроллеров включают Kong Ingress Controller (https://github.com/Kong/kubernetes-ingress-controller), Traefik Kubernetes Ingress Controller (https://mng.bz/OZgE) и Tyk Operator (https://github.com/TykTechnologies/tyk-operator).

https://liveinternet.club

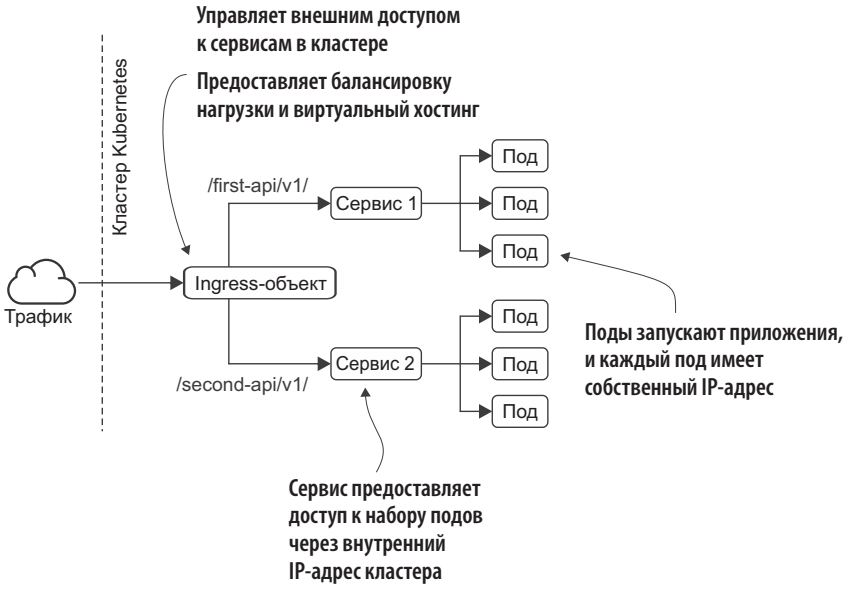


Рис. Д.3. Ingress-объект и сервис маршрутизируют трафик к подам

Ранее я отмечал, что менеджер контроллера обеспечивает выполнение контроллеров. Контроллер выполняет незавершаемый цикл управления, который наблюдает за общим состоянием объекта кластера и вносит изменения для обеспечения соответствия текущего состояния объекта желаемому, как показано на рис. Д.4. Каждый объект содержит *поле спецификации*, определяющее его желаемое состояние, и поле *статуса*, описывающее текущее состояние.

Объекты Kubernetes — поды, развертывания, сервисы, контроллеры и т. д. — представляют собой долгосрочные сущности в системе Kubernetes, отражающие состояние кластера. Создавая объект Kubernetes, вы описываете желаемое состояние кластера.

Объекты в Kubernetes можно создавать при помощи kubectl или напрямую с применением Kubernetes REST API. При использовании kubectl информация объекта предоставляется в файле, который называется *манифестом*. Файлы манифеста записываются в формате YAML (или JSON). Файлы манифеста содержат четыре обязательных поля:

- `apiVersion` — задает версию Kubernetes API;
- `kind` — описывает тип объекта Kubernetes, создаваемого на базе манифеста;
- `metadata` — информация, которая помогает однозначно идентифицировать объект (например, его имя или пространство имен);
- `spec` — описывает желаемое состояние объекта.

https://liveinternet.club

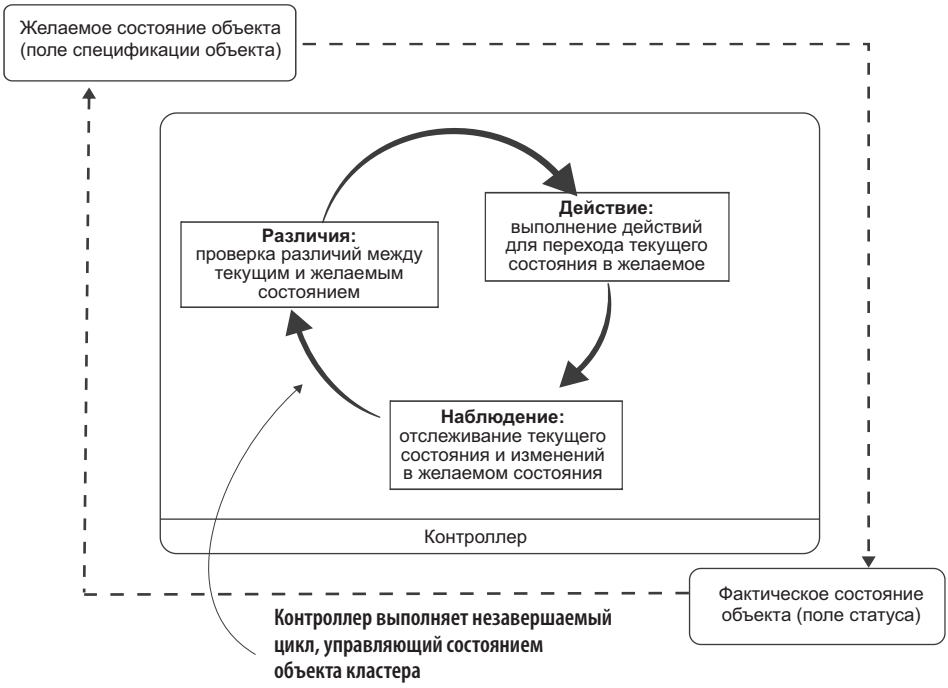


Рис. Д.4. Контроллеры согласуют желательное и фактическое состояние объектов Kubernetes

Пример файла манифеста приведен в следующем фрагменте. Допустим, у вас имеется набор подов, которые ведут прослушивание на порте 8080 и помечены именем `product-api`. Следующий манифест описывает объект сервиса с именем `productsapi-svc`, который прослушивает трафик на порте 7070 и перенаправляет трафик на порт 8080 для любого пода с меткой `product-api`:

```
apiVersion: v1
kind: Service
metadata:
  name: productsapi-svc
spec:
  ports:
    - port: 7070
      protocol: TCP
      targetPort: 8080
  selector:
    app: product-api
```

Если этот манифест хранится в файле `service.yaml`, он может быть применен к кластеру командой kubectl:

```
kubectl apply -f service.yaml
```

https://liveinternet.club

На этом мы завершаем краткое введение в Kubernetes, которое подготовит вас к главе 9 книги. Более подробное описание Kubernetes см. в книге «Kubernetes in Action» Марко Лукша (Marko Lukša) и Кевина Коннера (Kevin Conner) (2nd ed., Manning, 2020, https://mng.bz/Y7jA).

Д.5. Установка kubectl

Выше я упоминал kubectl (https://kubernetes.io/docs/reference/kubectl) — инструмент командной строки для управления Kubernetes. Команды kubectl обычно записываются в форме `kubectl <действие> <ресурс>`.

Чтобы установить kubectl в macOS, используйте Homebrew и выполните следующую команду:

```
brew install kubectl
```

Чтобы установить kubectl в Windows, используйте Chocolatey и выполните команду:

```
choco install kubernetes-cli
```

Информацию об установке kubectl для вашей разновидности Linux см. на странице https://mng.bz/GZnO.

Для проверки правильности установленной версии выполните команду:

```
kubectl version –client
```

Д.6. Установка локального кластера Kubernetes с использованием minikube

minikube (https://github.com/kubernetes/minikube) — простой способ начать использование Kubernetes для локальной разработки. Эта программа развертывает простой кластер Kubernetes, состоящий только из одного узла, на локальной машине. minikube работает в macOS, Linux и Windows. Прежде чем устанавливать minikube, убедитесь, что вы установили Docker.

Для macOS можно установить minikube с использованием Homebrew командой `brew install minikube`. Для Windows установите minikube с использованием менеджера пакетов Chocolatey командой `choco install minikube`. Инструкции по установке minikube для вашей разновидности Linux см. на странице https://minikube.sigs.k8s.io/docs/start.

После установки запустите minikube командой `minikube start`. Теперь проверьте, что minikube работает — откройте дашборд minikube (пользовательский веб-интерфейс Kubernetes со сводкой приложений, выполняемых в кластере). Выполните команду `minikube dashboard`, и дашборд откроется в вашем браузере.

https://liveinternet.club

Икенна Нвайву

Автоматизация доставки API. Улучшаем скорость и качество с APIOps и OpenAPI

Перевели с английского Е. Мансурова, Е. Матвеев

Научный редактор А. Белых

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*Н. Михеева*
Ведущий редактор	*Е. Строганова*
Литературный редактор	*М. Трусковская*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Т. Никифорова*
Верстка	*Е. Неволайнен*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 06.2026. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 28.05.26. Формат 70×100/16. Бумага офсетная. Усл. п. л. 33,540. Тираж 700. Заказ 0000.

https://liveinternet.club

КРОК

СОЗДАЕМ НАСТОЯЩЕЕ,
ИНТЕГРИРУЕМ БУДУЩЕЕ

croc.ru

КРОК — технологический партнер с комплексной экспертизой в области построения и развития инфраструктуры, внедрения информационных систем, разработки программных решений и сервисной поддержки.

Центры компетенций КРОК фокусируются на ключевых отраслевых кластерах — промышленность, финансовый сектор, розничные продажи, муниципальное управление, спорт и культура.

Ежегодно сотни проектов КРОК становятся системообразующими для экономики и социально-культурной сферы.

https://liveinternet.club