

Инфраструктура как код

С примерами
на Python
и Terraform

Розмари Ванг



 MANNING

 ДМК
издательство

 + | ТІ Облако

Розмари Ванг

Инфраструктура как код

Infrastructure as Code, Patterns and Practices

WITH EXAMPLES IN PYTHON AND TERRAFORM

ROSEMARY WANG



MANNING
Shelter Island

Инфраструктура как код

ПАТТЕРНЫ И МЕТОДЫ РАЗРАБОТКИ
С ПРИМЕРАМИ НА PYTHON И TERRAFORM

РОЗМАРИ ВАНГ



Москва, 2025

УДК 004.7:004.42
ББК 32.972.5
В17

Ванг Р.

В17 Инфраструктура как код / пер. с англ. В. С. Яценкова. – М.: ДМК Пресс, 2025. – 484 с.: ил.

ISBN 978-5-93700-334-8

Инфраструктура как код – это набор методов и процессов для построения и обслуживания инфраструктуры с использованием скриптов, файлов конфигурации или языков программирования. Прочитав эту книгу, вы научитесь создавать объекты инфраструктуры с помощью Python и Terraform, а также уверенно обновлять инфраструктуру, не прерывая работу системы. Советы экспертов по стратегиям совместной работы в команде помогут вам избежать нестабильности, повысить безопасность системы и сократить расходы.

Книга предназначена специалистам по инфраструктуре и профессиональным разработчикам, знакомым с языком Python, инструментами конфигурирования и публичными облачными сервисами.

УДК 004.7:004.42
ББК 32.972.5

Copyright © DMK Press 2025. Authorized translation of the English edition.

Copyright © 2022 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-6172-98229-5 (англ.)
ISBN 978-5-93700-334-8 (рус.)

© Manning Publications, 2022
© Перевод, оформление, издание, ДМК Пресс, 2025

Оглавление

https://t.me/it_boooks/2

- Часть I ПЕРВЫЕ ШАГИ 24**
 - 1 ■ Знакомство с понятием IaC..... 25
 - 2 ■ Разработка инфраструктуры как кода..... 57
 - 3 ■ Паттерны построения модулей инфраструктуры..... 87
 - 4 ■ Паттерны инфраструктурных зависимостей.....118
- Часть II МАСШТАБИРОВАНИЕ ДЛЯ РАБОТЫ В КОМАНДЕ 154**
 - 5 ■ Структурирование и совместное использование модулей 155
 - 6 ■ Тестирование 182
 - 7 ■ Модели непрерывной поставки и ветвления 222
 - 8 ■ Безопасность и комплаенс 261
- Часть III УПРАВЛЕНИЕ СЛОЖНОСТЬЮ ПРОИЗВОДСТВЕННОЙ ИНФРАСТРУКТУРЫ.....288**
 - 9 ■ Развертывание обновлений.....289
 - 10 ■ Рефакторинг.....326
 - 11 ■ Исправление ошибок.....361
 - 12 ■ Стоимость облачных услуг 382
 - 13 ■ Выбор и обновление инструментов.....421
 - 14 ■ Дополнительная глава от T1 Облако. О нашем опыте работы с IaC 446

Содержание

Приветствие генерального директора T1 Облако	11
Предисловие	13
Благодарности	15
Об этой книге	17
Об авторе	22
Иллюстрация на обложке	23

Часть I	ПЕРВЫЕ ШАГИ	24

1	Знакомство с понятием IaC	25
1.1	Что такое инфраструктура?	27
1.2	Что такое инфраструктура как код?	30
1.2.1	Ручная настройка инфраструктуры	30
1.2.2	Инфраструктура как код	31
1.2.3	Что не относится к IaC?	34
1.3	Ключевые принципы IaC	35
1.3.1	Воспроизводимость	35
1.3.2	Идемпотентность	37
1.3.3	Компонуемость	39
1.3.4	Развиваемость	41
1.3.5	Применение принципов разработки	43
1.4	Кто и зачем использует IaC?	44
1.4.1	Управление изменениями	44
1.4.2	Окупаемость затрат времени	45
1.4.3	Обмен знаниями	47
1.4.4	Безопасность	48
1.5	Инструменты	49
1.5.1	Примеры в этой книге	49
1.5.2	Инициализация инфраструктуры (provisioning)	52
1.5.3	Управление конфигурацией	53
1.5.4	Создание образа	54
	Краткое содержание главы	55

2	Разработка инфраструктуры как кода	57
2.1	Описание изменений инфраструктуры	59
2.2	Что такое неизменность инфраструктуры?	61
2.2.1	Внесение нештатных изменений	64
2.2.2	Переход к технологии IaC	66
2.3	Написание чистого кода	74

2.3.1	Передача контекста в систему контроля версий	74
2.3.2	Линтинг и форматирование	76
2.3.3	Именованние ресурсов	77
2.3.4	Переменные и константы	78
2.3.5	Параметризация зависимостей.....	79
2.3.6	Хранение секретов.....	84
	Краткое содержание главы.....	85

3	Паттерны построения модулей инфраструктуры	87
3.1	Паттерн «одиночка»	88
3.2	Паттерн «компоновщик»	92
3.3	Фабричный паттерн	96
3.4	Паттерн «прототип»	100
3.5	Паттерн «строитель»	105
3.6	Выбор паттерна	112
	Краткое содержание главы.....	116

4	Паттерны инфраструктурных зависимостей	118
4.1	Однонаправленные отношения	119
4.2	Внедрение зависимостей.....	121
4.2.1	Инверсия управления	121
4.2.2	Инверсия зависимостей	124
4.2.3	Применение внедрения зависимостей	131
4.3	Паттерн «фасад»	135
4.4	Паттерн «адаптер».....	139
4.5	Паттерн «медиатор»	146
4.6	Выбор паттерна	150
	Краткое содержание главы.....	152

Часть II **МАСШТАБИРОВАНИЕ ДЛЯ РАБОТЫ В КОМАНДЕ**

5	Структурирование и совместное использование модулей	155
5.1	Структура репозитория.....	156
5.1.1	Единый репозиторий.....	158
5.1.2	Раздельные репозитории	161
5.1.3	Выбор структуры репозитория	165
5.2	Управление версиями	170
5.3	Релиз.....	174
5.4	Совместное использование модулей	177
	Краткое содержание главы.....	180

6	Тестирование.....	182
6.1	Цикл тестирования инфраструктуры.....	185
6.1.1	Статический анализ	185
6.1.2	Динамический анализ	186
6.1.3	Среды тестирования инфраструктуры	188
6.2	Модульные тесты.....	189
6.2.1	Тестирование конфигурации инфраструктуры	191

6.2.2	Тестирование с помощью предметно-ориентированных языков.....	193
6.2.3	Когда следует применять модульные тесты?	197
6.3	Контрактные тесты.....	199
6.4	Интеграционные тесты.....	202
6.4.1	Тестирование модулей.....	202
6.4.2	Тестирование конфигурации среды	205
6.4.3	Проблемы тестирования.....	207
6.5	Сквозные тесты	209
6.6	Другие тесты	213
6.7	Выбор тестов.....	214
6.7.1	Стратегия модульного тестирования	216
6.7.2	Стратегия тестирования конфигурации.....	217
6.7.3	Как составить набор наиболее полезных тестов.....	219
	Краткое содержание главы.....	221

7	Модели непрерывной поставки и ветвления	222
7.1	Поставка изменений в производство	224
7.1.1	Непрерывная интеграция	225
7.1.2	Непрерывная поставка	227
7.1.3	Непрерывное развертывание.....	229
7.1.4	Выбор способа поставки	230
7.1.5	Модули	236
7.2	Модели ветвления	238
7.2.1	Функционально ориентированная разработка	239
7.2.2	Магистрально ориентированная разработка.....	244
7.2.3	Выбор модели ветвления	247
7.3	Внешняя проверка.....	252
7.4	GitOps	257
	Краткое содержание главы.....	259

8	Безопасность и комплаенс	261
8.1	Управление доступом и секретами	262
8.1.1	Принцип наименьших привилегий.....	263
8.1.2	Защита секретов в конфигурации	266
8.2	Инфраструктура тегирования	268
8.3	Политика как код.....	272
8.3.1	Механизмы и стандарты политики	273
8.3.2	Тесты безопасности	276
8.3.3	Тесты политики	279
8.3.4	Паттерны и методики.....	281
	Краткое содержание главы.....	287

Часть III **УПРАВЛЕНИЕ СЛОЖНОСТЬЮ ПРОИЗВОДСТВЕННОЙ ИНФРАСТРУКТУРЫ.....288**

9	Развертывание обновлений.....	289
9.1	Подготовка к изменениям	291
9.1.1	Наличие четкого плана действий.....	292

9.1.2	Повышение надежности	293
9.2	Blue-green развертывание	299
9.2.1	Развертывание зеленой инфраструктуры.....	302
9.2.2	Развертывание высокоуровневых зависимостей в зеленой инфраструктуре.....	304
9.2.3	Использование канареечного развертывания в зеленой инфраструктуре.....	307
9.2.4	Проведение регрессионного тестирования	314
9.2.5	Удаление синей инфраструктуры	316
9.2.6	Несколько примечаний	318
9.3	Инфраструктура, хранящая данные	319
9.3.1	Blue-green развертывание	319
9.3.2	Конвейер поставки обновлений	321
9.3.3	Канареечное развертывание	322
	Краткое содержание главы.....	324

10	Рефакторинг	326
10.1	Минимизация последствий рефакторинга	327
10.1.1	Уменьшение потенциального ущерба с помощью стратегии последовательного обновления (rolling update)	328
10.1.2	Поэтапный рефакторинг с использованием флагов функций (feature flags)	330
10.2	Декомпозиция монолитных конфигураций.....	338
10.2.1	Рефакторинг ресурсов высокого уровня	339
10.2.2	Рефакторинг ресурсов с зависимостями	352
10.2.3	Повторение процесса рефакторинга	356
	Краткое содержание главы.....	359

11	Исправление ошибок.....	361
11.1	Восстановление функциональности	362
11.1.1	Отмена изменений путем наката.....	363
11.1.2	Накат новых изменений	365
11.2	Устранение неполадок	366
11.2.1	Проверка наличия конфигурационного сдвига	366
11.2.2	Проверка зависимостей.....	369
11.2.3	Проверка различий между средами.....	371
11.3	Устранение причины сбоя	373
11.3.1	Устранение расхождений.....	374
11.3.2	Согласование различий в средах	377
11.3.3	Внесение изначально необходимого изменения	378
	Краткое содержание главы.....	381

12	Стоимость облачных услуг	382
12.1	Управление факторами затрат	383
12.1.1	Способы тестирования для контроля затрат.....	386
12.1.2	Автоматизация оценки затрат.....	390
12.2	Сокращение объема облачных излишков	399
12.2.1	Остановка непомеченных или неиспользуемых ресурсов	400
12.2.2	Запуск и остановка ресурсов по расписанию	401

12.2.3	Выбор правильного типа и размера ресурса	403
12.2.4	Применение автомасштабирования	404
12.2.5	Установка тега истечения срока действия ресурса	408
12.3	Оптимизация затрат	412
12.3.1	Создание сред по требованию	413
12.3.2	Использование нескольких облаков	414
12.3.3	Оценка трафика между регионами и облаками	415
12.3.4	Тестирование в производстве	417
	Краткое содержание главы.....	419

13	Выбор и обновление инструментов	421
13.1	Использование инструментов и модулей с открытым исходным кодом.....	422
13.1.1	Функциональность.....	423
13.1.2	Безопасность.....	425
13.1.3	Жизненный цикл.....	427
13.2	Обновление инструментов	429
13.2.1	План действий перед обновлением	430
13.2.2	Обратная совместимость	431
13.2.3	Критические изменения в новых версиях.....	434
13.3	Замена инструментов	437
13.3.1	Если новый инструмент поддерживает импорт	438
13.3.2	Если в новом инструменте нет функции импорта	438
13.4	IaC, управляемая событиями.....	442
	Краткое содержание главы.....	444

14	Дополнительная глава от T1 Облако. О нашем опыте работы с IaC	446
14.1	О Terraform-провайдере T1 Облако.....	446
14.2	О сценариях использования Terraform-провайдера у клиентов	447
14.3	О развитии Terraform-провайдера T1 Облако	448
14.4	Цифры и прогнозы	448
14.5	Примеры кода.....	448
14.5.1	Пример 1: создание виртуальной машины	449
14.5.2	Пример 2: управление ролями.....	452
14.5.3	Пример 3: создание кластера Kubernetes	453
14.6	Заключение	456
	Приложение А. Работа с примерами кода.....	457
	Приложение В. Решения к упражнениям	472
	Предметный указатель.....	480

Приветствие генерального директора T1 Облако

Друзья!

Давайте зададим вопрос: почему возник такой подход к управлению инфраструктурой, как Infrastructure as Code (IaC)? С развитием бизнеса растет и «зоопарк» информационных систем: серверов, баз данных, хранилищ, сетей, появляются серверы с уникальным набором установленного ПО. Управлять ими вручную не только тяжело – это занимает все больше времени, увеличивает риск влияния человеческого фактора, сокращает ваши возможности контролировать процесс изменений и, как следствие, вызывает критичные последствия для людей и бизнеса.

Как генеральный директор российского облачного провайдера, я сам свидетель и активный участник эволюции подходов к управлению инфраструктурой. Сегодня мы наблюдаем переход от традиционных ручных методов к более гибким и масштабируемым практикам.

Автор книги Розмари Ванг прошла этот путь. В книге она делится личным опытом, как стала использовать публичное облако, и объясняет, как пришла к применению инструментов IaC для управления инфраструктурой. Автор умело соединяет теоретический материал с практическими примерами. В частности, рассказывает о группировке ресурсов в модули, показывает, как ослабить и разделить инфраструктурные зависимости, рассуждает о выборе подходящих

инструментов. Еще одно важное преимущество книги: в ней не только описаны технические аспекты, но и представлен широкий взгляд автора на инфраструктуру как код. Например, Розмари Ванг уделяет внимание применению мер безопасности и комплаенса, которые необходимо реализовывать в инфраструктуре в процессе разработки.

В дополнительной главе книги наша команда тоже поделится опытом управления облачной инфраструктурой. Расскажет о реальных сценариях использования Terraform-провайдера и продемонстрирует примеры кода. Все это поможет читателю лучше понять, как можно упростить автоматизацию процессов управления инфраструктурой и другими ресурсами в облаке.

Эта книга пригодится ИТ-специалистам – системным и сетевым администраторам, инженерам по разработке, инженерам DevOps, архитекторам инфраструктуры, которые хотят как познакомиться с подходом IaC, так и глубже изучить его принципы. Книга «Инфраструктура как код» будет полезна и для тимлидов, специалистов по обеспечению надежности ИС, руководителей продуктовых направлений и всех, кто заинтересован в создании надежной и безопасной инфраструктуры.

Представленный материал подскажет, как упростить и улучшить управление инфраструктурой при помощи кода. Описанный подход IaC призван помочь как отдельным командам, так и всему бизнесу работать более эффективно и с меньшими рисками.

Надеюсь, эта книга станет вашим верным спутником в мире Infrastructure as Code: покажет возможности и перспективы, вдохновит на изменения и улучшения в ИТ.

Уверен, в будущем подход Infrastructure as Code будет играть все более важную роль в создании и управлении современными ИТ-системами.

В заключение хочу сказать большое спасибо всем, кто работал над этой книгой. Без вашего вклада она не стала бы такой, какой ее видит читатель. Отдельное спасибо за скрупулезную работу над выпуском книги хочу сказать Андрею Никитину, Алану Миллеру, Ивану Дьяченко, Павлу Иванникову, Валерию Ишутину, Елизавете Титаренко, Юлии Брагировой и Алине Окуневой. Хочу также поблагодарить переводчика и заместителя главного редактора издательства «ДМК Пресс» Валерия Яценкова за проделанный труд по переводу, адаптации книги и экспертную поддержку.

Антон Степанов,
генеральный директор
Т1 Облако

Предисловие

Когда я впервые пришла в дата-центр, он впечатлил меня сканером сетчатки на входе, мигающими огнями, гулом систем охлаждения и жгутами разноцветных проводов. Образование и опыт работы инженером-электронщиком позволили мне оценить сложность управления оборудованием. Но когда компания наняла меня для управления частной облачной платформой, я столкнулась с запутанной технологией облачных вычислений. Я больше не подключала провода и не монтировала серверы в стойки. Вместо этого я смотрела на индикаторы выполнения в пользовательском интерфейсе тысяч серверов и писала ужасные скрипты для их подготовки.

В тот момент я поняла, что нуждаюсь в новых знаниях. Я хотела максимально автоматизировать работу с инфраструктурой и научиться писать более надежный код, который могли бы использовать другие члены команды. Мой путь обучения отражал развитие облачных вычислений и философии DevOps. Нам нужно было научиться изменять и масштабировать инфраструктуру компании, чтобы поддерживать инновации в бизнесе и не затрагивать критически важные системы. Возможность простого получения инфраструктурных ресурсов по требованию заложена в идеологию общедоступных облачных сервисов, поэтому мы можем рассматривать доступную инфраструктуру как расширение своего программного обеспечения.

Я прошла тернистый путь обучения, став специалистом широкого профиля. Я рассчитала стоимость миграции в общедоступное облако в сотрудничестве со старшими разработчиками Java (задача, которая заставила меня плакать), применила паттерны проектирования и теорию разработки программного обеспечения к коду, опробовала методологию Agile и задала много вопросов специалистам по обеспечению качества и безопасности. По мере знакомства с различны-

ми точками зрения и приобретения технического опыта я пыталась помочь другим людям в качестве консультанта и в конечном итоге разработчика, поддерживающего внедрение инструментов управления инфраструктурой с открытым исходным кодом.

Я решила написать эту книгу, потому что часто слышала от системных администраторов, специалистов по безопасности и разработчиков программного обеспечения, что они хотят изучить методологию «инфраструктура как код» (infrastructure as code, IaC) и нуждаются в ресурсе, который объединяет паттерны и передовые методы разработки кода. В этой книге отражено все, что я сама хотела узнать об IaC, а также о соображениях и проблемах применения определенных паттернов и практик, вне зависимости от конкретных инструментов и технологий.

Я не ожидала, что в книге будет столько технической информации. Каждый раз, когда я показывала коллегам новую главу, мне говорили, что я о чем-то забыла, или рекомендовали выделить какую-то тему в отдельную главу. Многие главы касаются тем, которым посвящены целые книги (или даже документальные фильмы), но в этой книге они рассматриваются обобщенно и на высоком уровне. Я фокусируюсь только на самых важных вещах, которые вам необходимо знать, чтобы применить эту тему к IaC.

Вы можете взглянуть на примеры в этой книге и спросить: «Почему бы не использовать другой инструмент?» Я приложила максимум усилий, чтобы сбалансировать теории высокого уровня с практическими примерами. Листинги кода вызвали оживленные дискуссии среди моих рецензентов и редакторов, многие из которых предлагали расширения или замены, основанные на другом языке, инструменте и платформе! Но здесь надо иметь в виду, что я старалась найти комбинацию языков, инструментов и платформ для демонстрации важных закономерностей. На момент публикации книги вы найдете в репозитории GitHub примеры кода, написанные на Python, развернутые с помощью HashiCorp Terraform и запущенные на Google Cloud Platform (GCP). Каждый листинг сопровождается обобщенным описанием паттернов и методик, которые вы можете применять независимо от языка, инструмента или платформы.

Я надеюсь, что вы прочитаете эту книгу и найдете один или два приема, которые помогут вам написать более чистый код IaC, совместно работать над IaC в команде, а также масштабировать IaC до уровня всей вашей компании. Пожалуйста, не ждите, что вам пригодятся все паттерны и подходы или что вы будете применять их все одновременно. Но это не повод для разочарования! Я рассчитываю, что время от времени вы будете возвращаться к этой книге и находить еще несколько подходящих решений для своих проблем с IaC.

Благодарности

Мне исключительно повезло оказаться в окружении коллег и единомышленников, без помощи и поддержки которых не появилась бы эта книга.

Спасибо моему спутнику жизни Адаму, который помог мне найти время (и много литров кофе), чтобы сосредоточиться и поработать над этой книгой. Спасибо моей семье за стимулы продолжать изучение методов управления инфраструктурой. Вы внимательно слушали меня и говорили слова поддержки, даже если ничего не понимали в технических проблемах, с которыми я пыталась разобраться.

Я очень благодарна редакторам издательства Manning – Крису Филипсу, Майку Шепарду, Трише Лувар и Фрэнсису Лефковиц – за их терпение, поддержку, наставления и рекомендации. Мне очень приятно, что вы проявляете столько внимания и доброжелательной настойчивости по отношению даже к самым первым черновым наброскам. Я также хочу поблагодарить команду, работающую над созданием и продвижением этой книги.

Большое спасибо рецензентам, которые прочитали мою рукопись и нашли время и силы, чтобы дать мне отзывы и рекомендации: Козимо Аттанаси, Дэвид Криф, Дениз Вехби, Доминго Салазар, Эрнесто Карденас Кангауала, Джордж Хейнс, Гуалтьеро Теста, Джеффри Чу, Джереми Брайан, Джоэл Клермонт, Джон Гатри, Люсиан Мали, Майкл Брайт, Огнян Димитров, Питер Шотт, Рави Тамири, Шон Букер, Стэнфорд Гиллори, Стеффен Вейткамп, Стивен Оксли, Сильвен Мартель и Зорозайи Мукуя – благодаря вам эта книга стала намного лучше.

Спасибо членам сообщества HashiCorp за то, что высказали свое мнение, обсудили идеи и поддержали мое стремление написать эту книгу. Вы все вдохновили меня продолжать писать, несмотря на

технические сложности и периодические приступы синдрома самозванца. Отдельная благодарность – моему коллеге и техническому корректору Тейлору Долезалу, который смело погрузился в мой код и прочитал книгу в различных итерациях.

Я благодарна системным администраторам, тестировщикам, менеджерам по продуктам, инженерам инфраструктуры, бизнес-аналитикам, разработчикам программного обеспечения и инженерам по безопасности, с которыми мне довелось работать. Возможно, вы узнаете в этой книге некоторые паттерны и приемы, которые мы с вами обсуждали. Спасибо за все, чему вы меня научили. Вы являетесь частью сообщества, которое помогло этой книге появиться на свет.

Об этой книге

Я написала эту книгу, чтобы научить вас разрабатывать сценарии IaC, которые обновляют ресурсы инфраструктуры, не затрагивая критические бизнес-системы. Основное внимание уделяется паттернам и практикам, которые вы можете применять индивидуально, в команде или компании, занимаясь разработкой и поддержкой инфраструктуры, а также предлагаются конкретные примеры, демонстрирующие реализацию.

Кому адресована эта книга

Эта книга предназначена для всех действующих или будущих специалистов (разработчиков ПО, инженеров по безопасности, инженеров по обеспечению качества или архитекторов инфраструктуры), которые начинают использовать облачную инфраструктуру и IaC и намерены масштабировать ее в своей команде или компании. Вы напишете код конфигурации IaC, запустите его вручную и создадите ресурсы в общедоступном облаке.

Однако дальше вы столкнетесь с проблемами сотрудничества при использовании IaC в вашей команде или компании. Вы должны устранить противоречия между членами своей команды и другими командами, вносящими изменения в инфраструктуру, а также удовлетворить запросы к функциональности, оставаясь в рамках нормативов и требований безопасности. Хотя во многих других публикациях IaC рассматривают в контексте конкретного инструмента, в этой книге обобщаются паттерны и методики, которые можно применять к различным вариантам использования инфраструктуры, инструментам и системам с учетом их будущего развития.

Структура и краткое содержание книги

Эта книга разбита на три части, в общей сложности состоящие из 13 глав.

Часть I знакомит читателя с понятием IaC и принципами разработки кода инфраструктуры.

- Глава 1 дает определение методики IaC, перечисляет ее преимущества и принципы. В ней отмечено, что книга содержит примеры на Python, выполняемые в HashiCorp Terraform и развернутые на Google Cloud Platform (GCP). В этой главе я также даю обзор инструментов и вариантов использования, с которыми вы столкнетесь при знакомстве с IaC.
- Глава 2 посвящена принципу неизменности и способам переноса существующих ресурсов инфраструктуры в IaC. В ней также рассмотрены методы написания чистого кода IaC.
- Глава 3 предлагает ознакомиться с паттернами разделения зависимостей и группировкой ресурсов инфраструктуры в модули. Описание каждого паттерна сопровождается примером и списком вариантов использования.
- Глава 4 рассказывает, как управлять зависимостями между ресурсами и модулями инфраструктуры и разделять их с помощью внедрения зависимостей и некоторых общих паттернов.

Часть II описывает совместную разработку и поддержку IaC в команде.

- Глава 5 рассказывает о соглашениях и методах представления IaC в различных репозиториях и их совместном использовании в вашей команде.
- Глава 6 описывает стратегию тестирования инфраструктуры. В ней рассмотрены различные типы тестов и способы их написания для IaC.
- Глава 7 рассказывает о применении непрерывной поставки в IaC. Она дает общее представление о моделях ветвления кода и о том, как ваша команда может использовать их для изменения инфраструктуры.
- Глава 8 описывает методы разработки безопасного и соответствующего требованиям кода IaC, включая тестирование и тегирование.

Часть III рассказывает о том, как управлять IaC в вашей компании.

- Глава 9 рассказывает о применении принципа неизменности к инфраструктуре, включая пример blue-green развертывания.
- Глава 10 посвящена рефакторингу IaC для улучшения удобства сопровождения и снижения риска неудачных изменений в одной кодовой базе.
- Глава 11 описывает откат IaC и внесение/отмену изменений в системе.

- Глава 12 предлагает применять IaC для снижения затрат на облачные вычисления и содержит пример оценки стоимости IaC.
- Глава 13 завершает книгу практическими рекомендациями по выбору и обновлению инструментов IaC.

Многие концепции на протяжении всей книги дополняют друг друга, и если вы ранее не имели дела с IaC, имеет смысл читать главы по порядку. Подготовленные читатели могут выбрать разделы, которые лучше всего подходят для решения их текущих задач.

В любом случае вы можете начать с главы 1 или приложения А, чтобы узнать, как разбирать и запускать примеры, а затем перейти к изучению конкретных тем. Приложение А содержит дополнительную информацию о библиотеках, инструментах и платформах, связанных с примерами, а в приложении В даны ответы на упражнения.

Исходный код примеров

Полную и регулярно обновляемую версию исходного кода можно найти на GitHub (<https://github.com/joatmon08/manning-book>). Описания инфраструктуры могут быть весьма многословными, поэтому из некоторых листингов кода на страницах книги для ясности исключены определения всей инфраструктуры. В главах со 2 по 12 листинги кода представляют собой концептуальные примеры.

В упомянутых листингах кода используется комбинация Python 3.9, HashiCorp Terraform 1.0 и Google Cloud Platform. Приложение содержит дополнительную информацию о том, как запускать примеры, а также ссылки на инструменты и библиотеки. Я регулярно обновляю исходный код на GitHub для минорных версий инструментов.

Эта книга содержит множество примеров исходного кода, представленного как в нумерованном, так и в обычном формате. В обоих случаях код напечатан шрифтом фиксированной ширины, чтобы отличать его от обычного текста. Иногда также применяется выделение жирным шрифтом, чтобы выделить часть кода, которая изменилась по сравнению с предыдущими шагами в главе, например когда новая функция добавлена к существующей строке кода.

Во многих случаях исходный код был переформатирован под типографские требования; мы добавили разрывы строк и переработали отступы, чтобы они соответствовали доступному пространству страницы в книге. В редких случаях даже этого было недостаточно, и листинги содержат маркеры продолжения строки (➡). Кроме того, комментарии к исходному коду часто удалены из листингов, если код описан в тексте. Многие листинги сопровождаются пояснениями к коду, раскрывающими важные идеи.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не по-

нравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

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

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

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

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Облачный провайдер

Мне было сложно решить, какой облачный провайдер выбрать для примеров. Хотя Amazon Web Services (AWS) или Microsoft Azure (Azure) наиболее часто упоминаются в других публикациях, для создания инфраструктуры на их основе необходимо подготовить много ресурсов. Например, прежде чем вы сможете использовать инфраструктуру, вам придется сконфигурировать сеть, подсети, таблицы маршрутизации, шлюзы и группы безопасности. Поэтому я решила использовать в качестве основного поставщика облачных услуг Google Cloud Platform (GCP), чтобы оптимизировать количество ресурсов, которые вам необходимо создать.

Хотя я использую для примеров только GCP, все понятия, процессы и рекомендации можно адаптировать к другим провайдерам облачных услуг.

Для читателей, которые предпочитают использовать AWS или Azure, каждый пример содержит информацию об эквивалентных услугах на этих двух платформах. Кроме того, некоторые примеры содержат эквиваленты в репозитории кода.

В главе 1 более подробно сказано о причинах, по которым я использую GCP, и о том, как адаптировать примеры для Azure и AWS. В приложении А вы найдете инструкции по настройке и запуску примеров на GCP, а также советы для пользователей Azure и AWS¹.

Другие онлайн-ресурсы

Изучите онлайн-ресурсы вашего конкретного инструмента IaC или поставщика инфраструктуры. Многие из них предоставляют примеры методик и паттернов кода, реализованных в их инструментах².

¹ Перевод этой книги на русский язык подготовлен при содействии российского облачного провайдера «Т1 Облако» (<https://t1-cloud.ru/>), входящего в многопрофильный холдинг Т1. Перечень услуг данного провайдера соответствует общепринятым мировым требованиям к развертыванию корпоративной инфраструктуры, и мы полагаем, что использование примеров и паттернов кода из книги не вызовет затруднений у читателей. В любом случае вы можете обратиться за помощью и рекомендациями по разработке или переносу инфраструктуры к техническим специалистам провайдера по адресу support@t1.cloud. – Прим. ред.

² Российским читателям мы рекомендуем ознакомиться с технической документацией облачного провайдера «Т1 Облако»: <https://t1-cloud.ru/docs>. – Прим. ред.

Об авторе



Розмари Ванг работает над устранением технических и культурных барьеров между потребностями предприятия в инфраструктуре, требованиями безопасности и разработкой приложений. Она любит решать трудные задачи и выступает как активный участник сообщества разработчиков, лектор, писатель и сторонник инструментов инфраструктуры с открытым исходным кодом. Когда Розмари не рисует маркером на доске, она отлаживает множество различных инфраструктурных систем на своем ноутбуке и между делом поливает комнатные растения.

Иллюстрация на обложке

Рисунок на обложке книги называется «Баядерка», что означает «индийская храмовая танцовщица». Он взят из коллекции Жака Грасе де Сен-Совер, опубликованной в 1797 году. Каждая иллюстрация в ней прорисована до мелких деталей и раскрашена вручную.

В те времена по одежде можно было легко определить, где живут люди и какое у них занятие или общественное положение. Издательство Manning стремится подчеркнуть изобретательность и инициативность компьютерного бизнеса с помощью иллюстраций, основанных на богатом разнообразии региональной культуры минувших веков, собранных в яркие коллекции.

Часть I

Первые шаги

Что такое *инфраструктура как код* (infrastructure as code, IaC) и как этот код писать? В первой части вы найдете ответ на эти вопросы и познакомитесь с методиками и паттернами, которые можно применить при написании IaC. В главе 1 вы узнаете, как работает IaC, какие проблемы решает этот подход и как эта книга поможет вам его реализовать. В главе 2 говорится о том, как писать чистый код, и даны рекомендации по определению IaC в существующих системах.

В главах 3 и 4 рассматриваются паттерны объявления инфраструктурных групп, называемых *модулями*, и их зависимости. Вы изучите паттерны модулей инфраструктуры и узнаете, как разделить зависимости модулей для поддержки изменений и минимизации ущерба, если что-то пойдет не так. В этих главах также предлагаются рекомендации по выбору наиболее подходящих паттернов для вашей ситуации и сферы деятельности.

Знакомство с понятием IaC

https://t.me/it_boooks/2

Основные темы главы:

- определение *инфраструктуры*;
- определение *инфраструктуры как кода*;
- почему важно рассматривать инфраструктуру как код.

Если вы только начали работать с поставщиками облачных услуг или инфраструктурой центров обработки данных, то наверняка ощутили небольшой страх перед объемом знаний, которые вам нужно усвоить. Ведь вы даже не знаете, что вам нужно знать, чтобы выполнять свою работу! Вам предстоит многое изучить, начиная с концепций инфраструктуры центров обработки данных, новых предложений общедоступных облаков, оркестровки контейнеров, языков программирования и заканчивая разработкой программного обеспечения.

Но, помимо изучения всего и сразу, вы также должны идти в ногу с требованиями вашей компании к инновациям и росту. Создание систем, способных обеспечить развитие предприятия в долгосрочной перспективе, – непростая задача. Вам нужен способ поддержки более сложных систем, минимизации усилий по обслуживанию и предотвращения сбоев в работе клиентов, использующих ваше приложение.

Что нужно знать для работы с облачными вычислениями или инфраструктурой центра обработки данных? Как масштабировать свои системы в рамках команды и вашей организации?

В основе ответа на оба вопроса лежит *инфраструктура как код* (IaC) – способ автоматизации изменений инфраструктуры в виде кода, удовлетворяющий требованиям масштабируемости, надежности и безопасности.

Использовать IaC может каждый: от системных администраторов, SRE-инженеров, инженеров DevOps, инженеров по безопасности и разработчиков программного обеспечения до инженеров по обеспечению качества. Независимо от того, читаете ли вы первое руководство по IaC или получили сертификат облачного провайдера, вы всегда сможете применить свои знания и навыки, чтобы упрощать, укреплять и масштабировать инфраструктуру.

Эта книга опирается на практический подход к IaC путем применения методик и приемов разработки ПО к управлению инфраструктурой. В книге представлены такие приемы, как тестирование, непрерывная поставка, рефакторинг и паттерны проектирования с учетом инфраструктуры. Вы найдете в ней практические примеры и паттерны кода, которые помогут вам управлять своей инфраструктурой независимо от применяемых средств автоматизации, инструментов, платформы или технологии.

Я разделила эту книгу на три части (рис. 1.1). В части I описаны методы, которые вы можете применить для разработки IaC, а в части II представлены паттерны и методы для совместной командной работы над ней. В части III рассматриваются некоторые подходы к масштабированию IaC в вашей организации.

Многие подходы и методики, описанные в этой книге, применимы ко всем трем областям. Ваша способность продумать и написать хорошую конфигурацию IaC помогает не только внести личный вклад в общее дело, но и масштабировать его до уровня команды и организации. Правильно написанная конфигурация помогает решить проблемы совместной работы над IaC, особенно по мере роста численности занятых в ней людей.

Часть I начинается со знакомства с понятием инфраструктуры и объяснения общих приемов проектирования IaC. Эта часть важна, потому что описывает основополагающие принципы, знание которых поможет вам масштабировать IaC в будущем. Даже если вы знакомы с IaC, я рекомендую не пропускать главы первой части, чтобы заложить надежную основу для более сложных концепций.

В частях II и III вы изучите паттерны и методы, необходимые для масштабирования систем и поддержки инфраструктуры критически важных приложений. Эти методы касаются вас лично, вашей команды и организации – от создания одного оповещения о метриках приложения до внесения изменений в сети организации со штатом в 50 000 человек. В этих частях многие термины и понятия дополняют друг друга, поэтому вам будет полезно читать главы по порядку.

В этой книге рассматриваются пересечения зон ответственности, относящихся к вам, вашей команде и организации в отношении IaC с целью масштабирования ваших ресурсов и систем и поддержки критически важных приложений

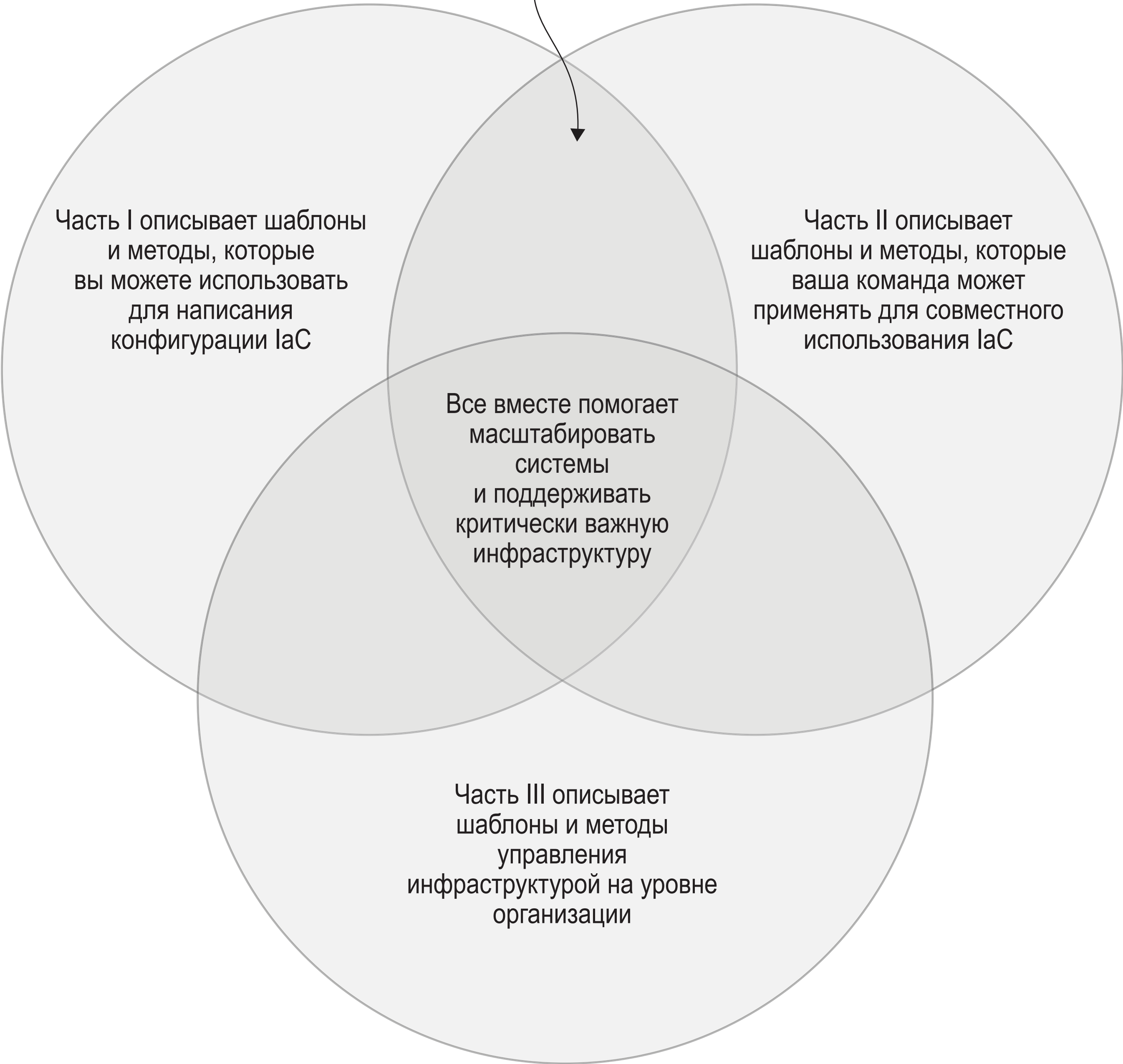


Рис. 1.1 В этой книге вы узнаете о пересечении различных областей ответственности за разработку инфраструктуры и о том, какую роль это играет для поддержки критически важных приложений

1.1 Что такое инфраструктура?

Прежде чем я углублюсь в IaC, давайте начнем с определения *инфраструктуры*. Когда я только начинала работать в центре обработки данных (ЦОД), инфраструктуру организации часто определяли как набор оборудования, обеспечивающего функции вычисления, передачи и хранения данных. На рис. 1.2 показано, как приложения запускаются на серверах (вычислительные ресурсы), подключаются через коммутатор (сеть передачи данных) и хранят данные на дисках (хранилище).

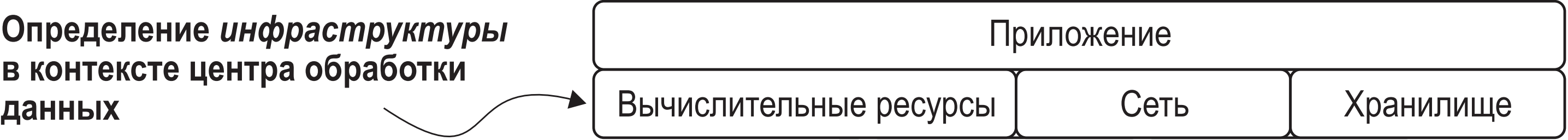


Рис. 1.2 Определение *инфраструктуры* в контексте центра обработки данных включает в себя сетевые и вычислительные ресурсы, а также ресурсы хранения, используемые для запуска приложения

Эти три категории соответствовали физическим устройствам, которыми мы управляли в центре обработки данных. Знаете, как мы вносили изменения в инфраструктуру? Мы сканировали наши пропуски, чтобы попасть в здание, подключались к серверу в стойке, вводили команды и надеялись, что после этого он продолжит работать. С появлением облачных вычислений мы продолжили использовать привычные категории для виртуализации конкретных устройств.

Однако определение инфраструктуры с точки зрения центра обработки данных не совсем применимо к сегодняшним услугам и технологиям. Представьте, что другая команда просит вас помочь доставить их приложение в промышленную среду для пользователей. Вы составляете контрольный список требований, который содержит следующие пункты:

- достаточное количество серверов;
- сетевое подключение для пользователей;
- база данных для хранения данных приложения.

Если все пункты контрольного списка выполнены, означает ли это, что команда сможет запустить свое приложение в промышленной среде? Не обязательно. Вы не знаете наперед, действительно ли вы настроили достаточное количество серверов или правильный доступ для входа в приложение. Вам также необходимо знать, влияет ли задержка сети на подключение к базе данных приложения.

В этом узком определении инфраструктуры вы упускаете из виду некоторые важные задачи, которые необходимо решить перед развертыванием приложения в промышленной среде, в том числе следующие:

- мониторинг метрик приложения;
- экспорт показателей для бизнес-отчетности;
- настройка оповещений для команд, работающих с приложением;
- добавление проверок работоспособности серверов и баз данных;
- поддержка аутентификации пользователей;
- регистрация и агрегирование событий приложений;
- хранение и смена паролей базы данных в менеджере секретов.

Эти задачи необходимо решить, чтобы развернутое в промышленной среде приложение работало надежно и безопасно. Вы можете рассматривать эти задачи как *эксплуатационные* требования, но они все равно требуют ресурсов инфраструктуры.

Помимо инфраструктуры, связанной с эксплуатацией, поставщики общедоступных облачных сервисов абстрагируют управление базовыми сетями, вычислениями и хранением и вместо отдельных ресурсов предлагают *платформу как услугу* (platform as a service, PaaS) – от объектных хранилищ, таких как S3, до платформ потоковой передачи событий, таких как Apache Kafka. Поставщики даже предлагают *функцию как услугу* (function as a service, FaaS) или *контейнеры как услугу* – это дополнительные абстракции для вычислительных

ресурсов. Существует растущий рынок *программного обеспечения как услуги* (software as a service, SaaS), такого как ПО для мониторинга производительности размещенных приложений. Эта услуга также может потребоваться для поддержки приложения в производстве, и ее можно рассматривать как часть инфраструктуры.

При таком разнообразии услуг мы не можем описать инфраструктуру только вычислительными ресурсами, сетями, хранилищами данных. Нам необходимо включить в поставку приложений операционную инфраструктуру, решения PaaS или SaaS. На рис. 1.3 модель инфраструктуры доработана и содержит дополнительные решения, такие как SaaS и PaaS, необходимые для развертывания и эксплуатации приложения.



Рис. 1.3 Инфраструктура приложения может включать очереди в общедоступном облаке, контейнеры, в которых выполняются приложения, бессерверные функции для дополнительной обработки или даже службы мониторинга для проверки работоспособности системы

Из-за постоянно растущей сложности, различия операционных моделей и пользовательской абстракции управления центром обработки данных мы не можем ограничить определение инфраструктуры аппаратными или физическими устройствами, связанными с вычислениями, сетью или хранилищем.

ОПРЕДЕЛЕНИЕ *Инфраструктура* – это программное обеспечение, платформа или оборудование, которые обеспечивают доставку или развертывание приложений на промышленном стенде.

Вот неполный перечень вариантов инфраструктуры, с которыми вы можете столкнуться:

- серверы;
- платформы оркестрации (например, Kubernetes, HashiCorp Nomad);
- сетевые коммутаторы;

- балансировщики нагрузки;
- базы данных;
- объектные хранилища;
- кеши;
- очереди;
- платформы потоковой передачи событий;
- платформы мониторинга;
- системы конвейеров данных;
- платежные платформы.

Расширенное определение инфраструктуры позволяет найти общий язык между командами, управляющими ресурсами для различных целей. Например, команда, управляющая структурой *непрерывной интеграции* (continuous integration, CI) организации, использует инфраструктуру либо SaaS непрерывной интеграции, либо вычислительные ресурсы из общедоступного облака. Другая команда в своей работе опирается на эту структуру, превращая ее, таким образом, в критически важную инфраструктуру.

1.2 Что такое инфраструктура как код?

Прежде чем рассматривать инфраструктуру как код, мы должны разобраться в ручной настройке инфраструктуры. В этом разделе я описываю проблему с ручной настройкой инфраструктуры. Затем я даю определение инфраструктуры как кода.

1.2.1 Ручная настройка инфраструктуры

В составе команды, обслуживающей сети, я научилась менять сетевой коммутатор, копируя и вставляя команды из текстового документа. Однажды я вставила команду shutdown (выключить) вместо no shutdown (не выключать) и отключила сетевой интерфейс! Я быстренько включила его обратно, надеясь, что никто не заметил мою оплошность и это ни на что не повлияло. Однако неделю спустя я обнаружила, что в тот момент отключилось соединение с критически важным приложением и это повлияло на несколько запросов клиентов.

Оглядываясь назад, я могу сказать, что столкнулась с несколькими проблемами при ручном копировании, вставке команд и настройке инфраструктуры. Во-первых, я понятия не имела, на какие ресурсы повлияет мое изменение (этот показатель называется *blast radius*, или *радиус поражения*). Я не знала, какие сети или приложения используют этот интерфейс.

ОПРЕДЕЛЕНИЕ Радиус поражения характеризует негативное влияние, которое неудачное изменение оказывает на систему. Большой радиус поражения часто затрагивает больше компонентов или наиболее важные компоненты.

Во-вторых, сетевой коммутатор выполнил мою команду, не проверив ее действие или назначение. Наконец, никто больше не знал, что повлияло на обработку запросов клиентов, и коллегам потребовалась неделя, чтобы определить основную причину – мою ошибочно скопированную команду.

Как описание инфраструктуры в виде кода поможет обнаружить мою ошибочно скопированную команду сетевого переключения? Я сохраняю команды конфигурации и автоматизации в виде сценария настройки инфраструктуры. Чтобы выявить свою ошибку, в дальнейшем я создаю виртуальный коммутатор и тест, который запускает мой сценарий и проверяет работоспособность интерфейса.

После успешного прохождения тестов я отправляю изменение в промышленную среду, поскольку тесты подтвердили правильность команды. Если я применю неправильную команду, я смогу выполнить поиск по конфигурации инфраструктуры, чтобы определить, какие приложения работают в затронутой сети. Вы можете обратиться к главе 6, где описаны методы тестирования, и к главе 11, где говорится об отмене изменений.

Помимо риска ошибочной конфигурации, ручная настройка иногда заметно замедляла скорость моей работы. Однажды мне потребовалось почти два месяца, чтобы протестировать свое приложение на базе данных. За эти два месяца моя команда отправила более 10 заявок, связанных с созданием базы данных, настройкой новой маршрутизации для подключения приложения к базе данных и редактированием правил брандмауэра, разрешающих мое приложение. Команда поддержки платформы получала заявки и вручную вносила все настройки в публичное облако. У команд разработчиков приложений не было прямого доступа к облаку из соображений безопасности.

Другими словами, ручная настройка инфраструктуры часто не масштабируется по мере роста сложности систем и численности команд разработчиков. *Ручные настройки увеличивают частоту неудачных изменений в системах, замедляют разработку и подвергают системы потенциальной атаке с помощью уязвимостей системы безопасности.* У вас всегда будет соблазн обновить некоторые значения в консоли. Однако эти изменения накапливаются.

Любое следующее изменение может внести в систему сбой, который будет трудно устранить, поскольку предыдущие изменения не были проверены или организованы. Такие изменения, как обновление настроек брандмауэра, чтобы разрешить отладочный трафик в процессе разработки, могут непреднамеренно сделать систему уязвимой для атак.

1.2.2 Инфраструктура как код

Как же нам обновлять инфраструктуру, если отказаться от ручных изменений? Вы можете применить традиционный цикл разработ-

ки ПО к ресурсам и конфигурации инфраструктуры в форме IaC. Но цикл разработки инфраструктуры выходит за рамки файлов конфигурации и сценариев.

Инфраструктура должна иметь возможность масштабироваться, реагировать на сбои, поддерживать быструю разработку ПО и обеспечивать безопасность приложений. Жизненный цикл разработки инфраструктуры включает в себя более специализированные приемы и методики для поддержки совместной работы, развертывания и тестирования. На рис. 1.4 показано, как упрощенный рабочий процесс меняет инфраструктуру с помощью конфигурации или сценариев и передает их в систему контроля версий. Событие коммита автоматически запускает рабочий процесс по развертыванию и тестированию изменений в вашей инфраструктуре.

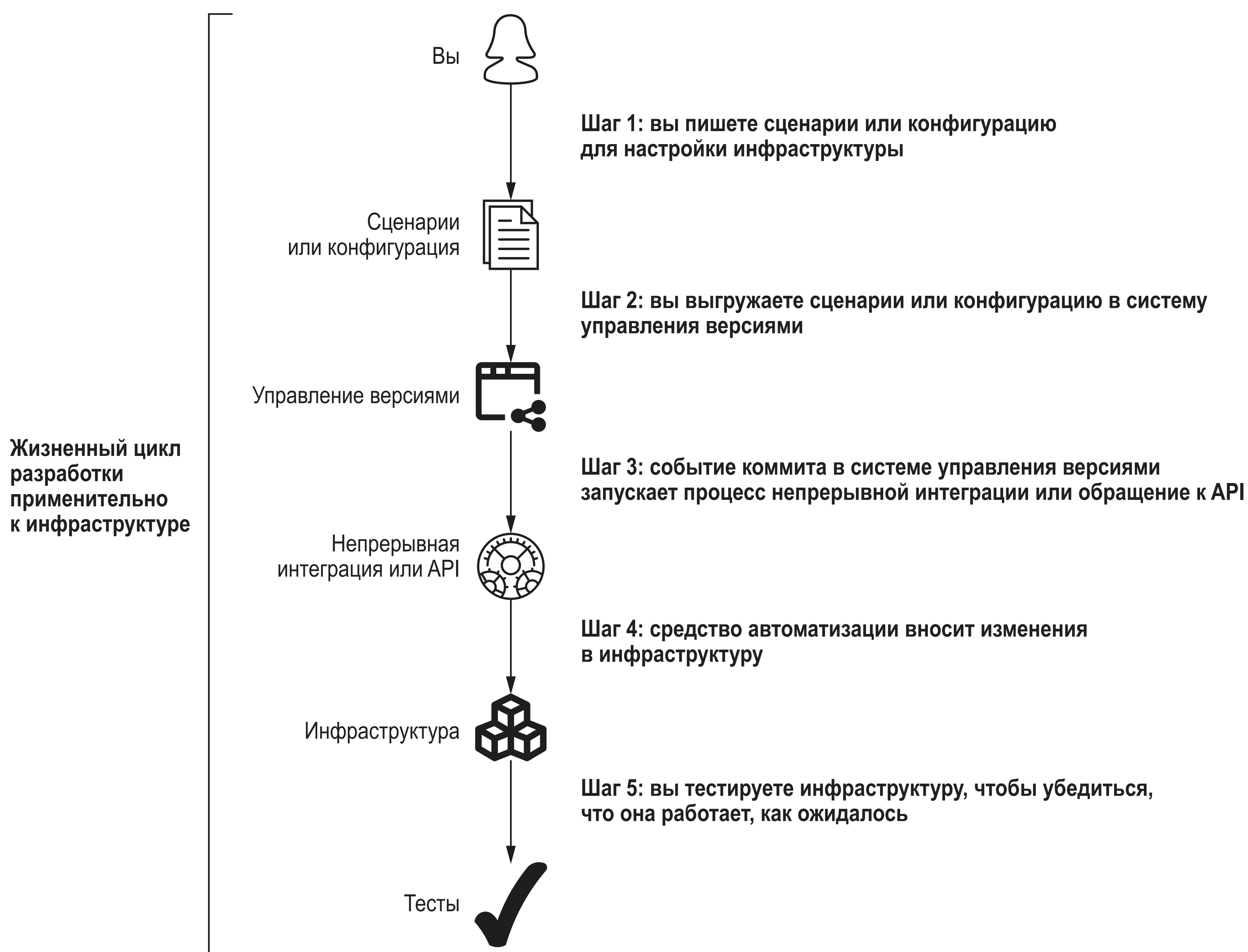


Рис. 1.4 Жизненный цикл разработки инфраструктуры включает в себя написание кода как документации (самодокументируемого кода)¹, передачу его в систему контроля версий, автоматическое применение к инфраструктуре и последующее тестирование

¹ Концепция *code as documentation* предусматривает разработку настолько качественного, строго структурированного и чистого кода, что он почти не нуждается в дополнительной документации для последующего понимания, использования и поддержки. Подробнее об этом можно прочитать в статье Мартина Фаулера: <https://martinfowler.com/bliki/CodeAsDocumentation.html>. – Прим. перев.

Почему мы уделяем такое внимание жизненному циклу разработки? Его можно взять за основу для управления изменениями и проверки того, не влияют ли они на вашу систему. Жизненный цикл представляет инфраструктуру в виде кода, который автоматизирует изменения инфраструктуры и применяет такие практики DevOps, как контроль версий и непрерывная поставка.

ОПРЕДЕЛЕНИЕ *Инфраструктура как код (IaC)* подразумевает применение методов DevOps для автоматизации изменений инфраструктуры в кодифицированной форме, чтобы обеспечить масштабируемость, отказоустойчивость и безопасность.

IaC часто называют обязательной практикой DevOps. Это, безусловно, относится к автоматизации модели CAMS (culture, automation, measurement, sharing – культура, автоматизация, измерение и совместное использование). На рис. 1.5 методика IaC рассматривается как часть философии и практики автоматизации в модели DevOps. Практика использования кода как документации, контроля версий, приемов разработки ПО и непрерывной поставки соответствует рабочему процессу разработки, который мы обсуждали ранее.

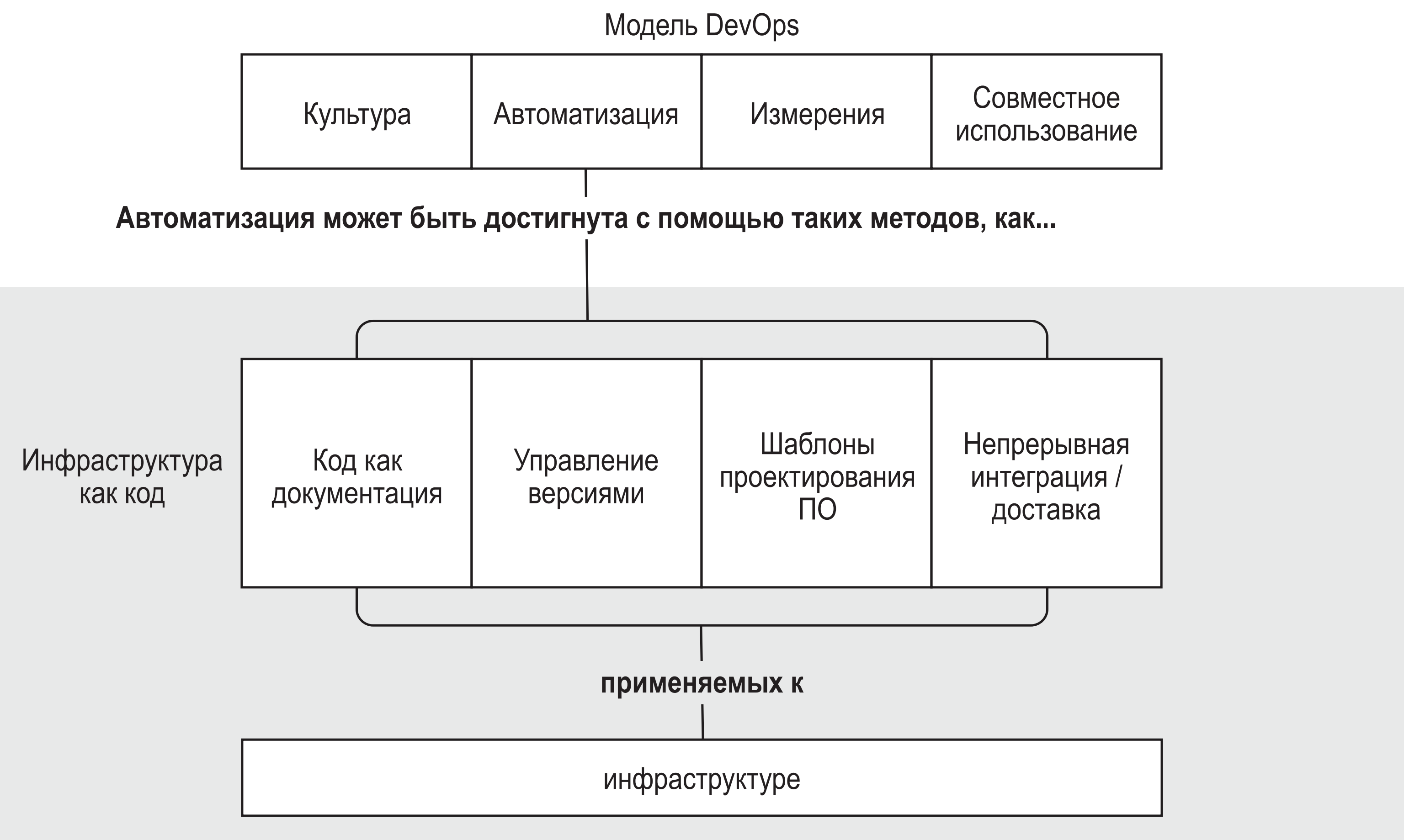


Рис. 1.5 IaC применяет к инфраструктуре контроль версий, паттерны программного обеспечения, непрерывную интеграцию и код как документацию

Но стоит ли ограничивать роль IaC только автоматизацией модели DevOps? Для использования IaC вашей организации не обязательно внедрять DevOps. Преимущества IaC облегчают внедрение и улучшают показатели DevOps, но при этом применимы к любой конфигурации инфраструктуры. Вы в любом случае можете использовать

методы IaC для улучшения процесса внесения изменений в инфраструктуру, не влияя на промышленную среду.

ПРИМЕЧАНИЕ В этой книге вы встретите упоминание некоторых методик DevOps, но я не уделяю особого внимания их теории или принципам. Для более глубокого понимания DevOps я рекомендую прочитать книгу *Accelerate* (Nicole Forsgren et al., IT Revolution Press, 2018). Вам также может пригодиться книга *The Phoenix Project* (Gene Kim et al., IT Revolution Press, 2013), в которой описана трансформация вымышленной организации, внедряющей DevOps.

В этой книге мы рассмотрим современные подходы к кодификации инфраструктуры, позволяющие решить проблему масштабирования, при одновременном сохранении надежности и безопасности инфраструктуры для пользователей приложений, независимо от того, используете вы собственный центр обработки данных или облако. Традиционные приемы разработки ПО, такие как контроль версий файлов конфигурации, конвейеры CI и тестирование, помогут вам масштабировать и развивать изменения в инфраструктуре, одновременно сокращая время простоя и создавая более безопасную конфигурацию.

1.2.3 Что не относится к IaC?

Если вы наберете текстовый документ с описанием конфигурации, будет ли это считаться IaC? Относится ли к методике IaC добавление инструкций по настройке в заявку на изменение инфраструктуры? Вы можете возразить, что руководство по настройке очереди или shell-скрипт для настройки сервера относится к IaC. Каждый из этих примеров относится к IaC, если вы можете использовать их для следующих целей:

- надежно и точно воспроизводить инфраструктуру, которая представлена в файле;
- возвращать конфигурацию к определенной версии или моменту времени;
- взаимодействовать с коллегами и оценивать радиус поражения при изменении ресурсов.

Однако конфигурации или сценарии обычно являются устаревшими, неверсионными или имеют неоднозначное назначение. Возможно, вам даже будет сложно понять и изменить конфигурацию, написанную в инструменте IaC. Этот инструмент облегчает рабочие процессы IaC, но не обязательно способствует методам и подходам, которые позволяют системам расти, одновременно снижая операционную ответственность и вероятность неудачных изменений. Вам следует знать набор принципов, определяющих IaC.

1.3 Ключевые принципы IaC

Как я уже упоминала, не каждый фрагмент кода или конфигурация, связанные с инфраструктурой, позволяют масштабировать систему или уменьшать время простоя. На протяжении всей книги я подчеркиваю, как принципы IaC применяются к определенным листингам кода или практикам. Вы даже можете использовать эти принципы для оценки вашего варианта IaC.

Хотя вы можете по своему желанию дополнять или сокращать список, я рекомендую запомнить четыре наиболее важных принципа под аббревиатурой RICE. Это *воспроизводимость* (reproducibility), *идемпотентность* (idempotency), *компонуемость* (composability) и *развиваемость* (evolvability). В следующих разделах я дам определения и расскажу о применении каждого принципа.

1.3.1 Воспроизводимость

Представьте, что кто-то просит вас создать среду разработки с очередью и сервером. Вы отправляете набор файлов конфигурации своим коллегам по команде. С помощью этих файлов они воспроизводят для себя новую среду менее чем за час. На рис. 1.6 показано, как вы поделились своей конфигурацией и помогли товарищам по команде воспроизвести новую среду. Вы воспользовались воспроизводимостью – первым принципом IaC!

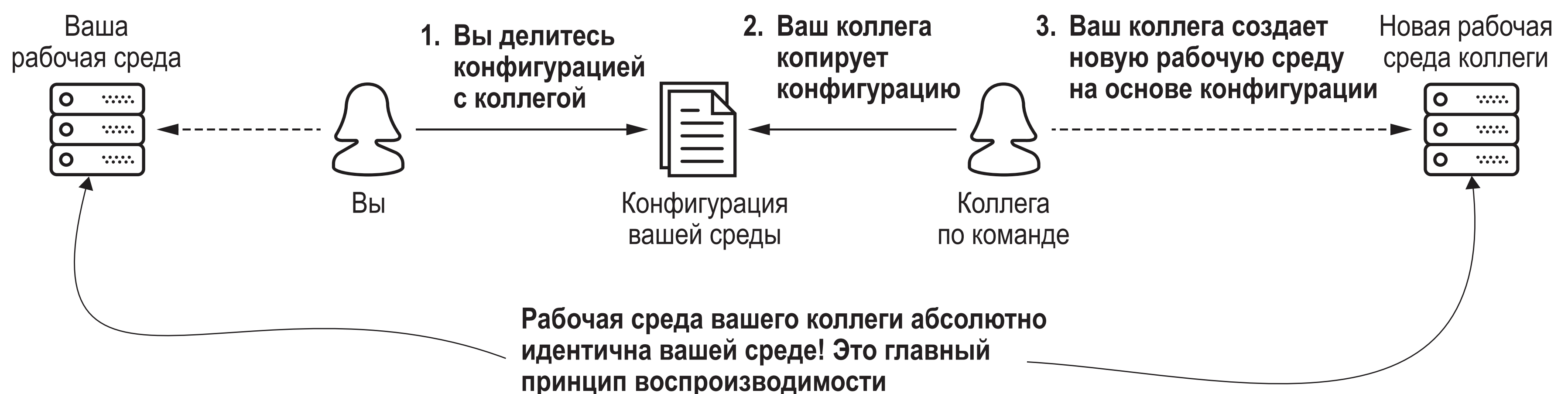


Рис. 1.6 Изменения, вносимые вручную, приводят к расхождению между системой контроля версий и фактическим состоянием и влияют на воспроизводимость, поэтому необходимо сохранять изменения в системе контроля версий

Почему подход IaC должен следовать принципу воспроизводимости? Возможность копировать и повторно использовать конфигурацию инфраструктуры экономит вам и вашей команде время на начальное проектирование. Вам не нужно изобретать велосипед, чтобы создавать новые среды или инфраструктурные ресурсы.

ОПРЕДЕЛЕНИЕ Принцип воспроизводимости означает, что вы можете использовать одну и ту же конфигурацию для многократного воспроизведения окружения или инфраструктуры.

На самом деле придерживаться принципа воспроизводимости сложнее, чем кажется. Он не ограничивается простым копированием и вставкой конфигурации. Чтобы ощутить нюансы, представьте, что вам нужно уменьшить адресное пространство сети с /16 до /24. У вас есть готовая конфигурация IaC, которая описывает сеть. Однако вы решаете выбрать простой путь: войти в консоль управления облачными службами и ввести /24 в текстовое поле.

Прежде чем войти в свою учетную запись облачного провайдера, вы размышляете о том, соответствует ли ваш рабочий процесс внесения изменений требованиям воспроизводимости. Вы задаете себе следующие вопросы:

- узнают ли коллеги по команде, что вы обновили сеть?
- если вы запустите сценарий конфигурации, вернется ли сетевое адресное пространство к /16?
- если вы создадите новую среду с конфигурацией в системе контроля версий, будет ли она иметь адресное пространство /24?

На каждый из этих вопросов вы отвечаете «нет». Вы не можете гарантировать, что ручное изменение будет успешно воспроизведено.

Если вы введете /24 в консоли облачного провайдера, ваша сеть отклонится от желаемого состояния, выраженного в IaC (рис. 1.7). Чтобы обеспечить воспроизводимость, вам придется обновить конфигурацию в системе контроля версий на /24 и применить инструмент автоматизации изменений.

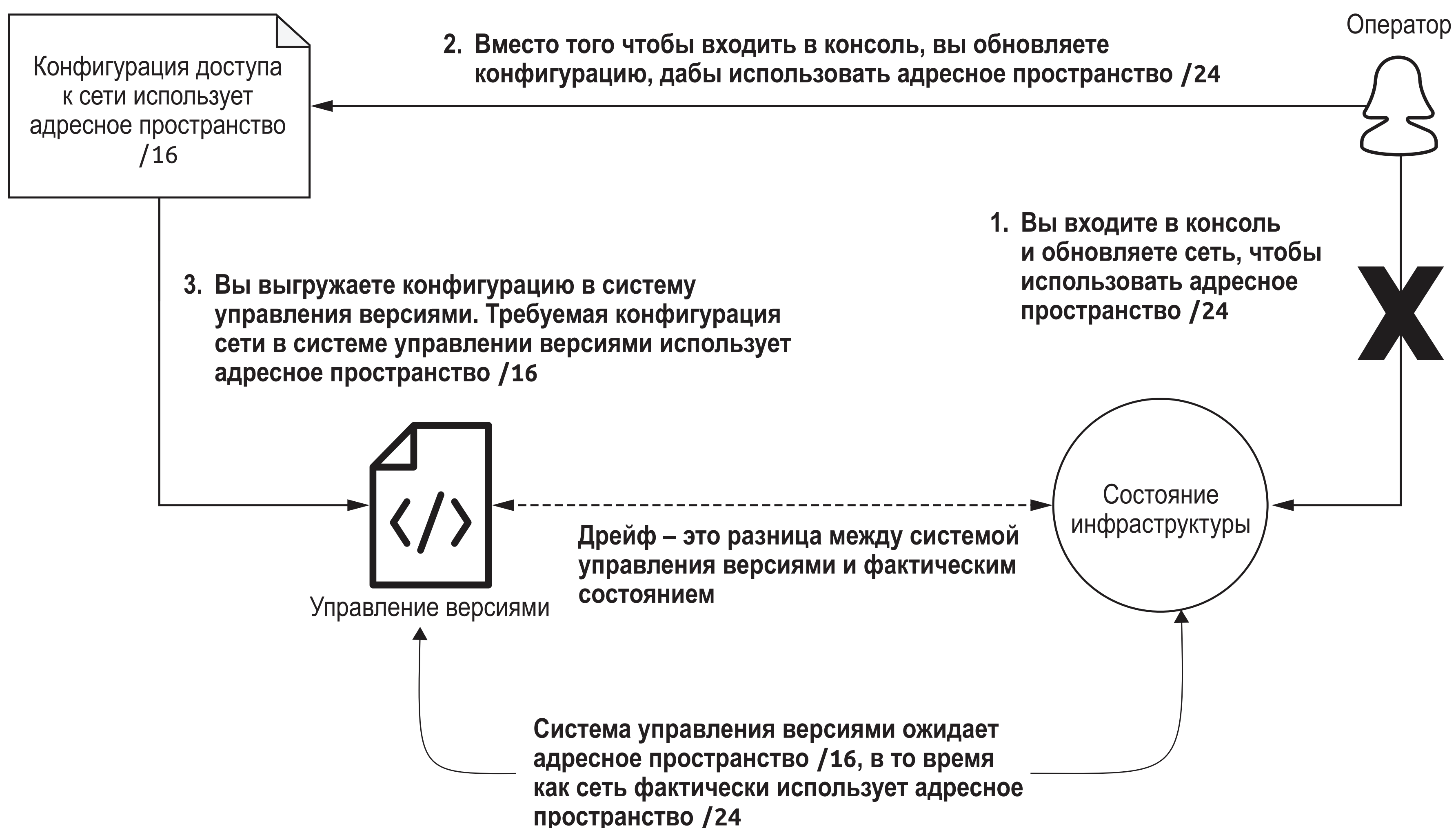


Рис. 1.7 Изменения, внесенные вручную, приводят к расхождению между конфигурацией, сохраненной в системе контроля версий, и фактическим состоянием и влияют на воспроизводимость, поэтому вы должны зафиксировать изменения в системе контроля версий

Этот сценарий демонстрирует проблему обеспечения воспроизводимости. Вам необходимо свести к минимуму несоответствие между ожидаемой и фактической конфигурациями инфраструктуры, также называемое дрейфом конфигурации.

ОПРЕДЕЛЕНИЕ *Дрейф конфигурации* – это смещение конфигурации инфраструктуры от ожидаемого состояния к фактическому.

На практике для соблюдения принципа воспроизводимости достаточно поместить файлы конфигурации в систему контроля версий и выполнять регулярные коммиты обновлений. Соблюдение принципа воспроизводимости облегчает командную работу и развертывание сред тестирования, аналогичных производственным.

В главе 6 вы узнаете больше о средах тестирования инфраструктуры, для которых особенно важна воспроизводимость. Вы также будете применять воспроизводимость к практикам и паттернам тестирования и обновления: от создания тестовой инфраструктуры, зеркально отображающей рабочую среду, до развертывания новой инфраструктуры для замены старых систем (blue-green развертывания).

1.3.2 Идемпотентность

Некоторые разработчики IaC ограничиваются лишь соблюдением принципа воспроизводимости, что означает получение стабильно одинаковых результатов при каждом запуске сценария автоматизации. Я утверждаю, что IaC нуждается в более строгих требованиях. Каждый запуск автоматизации должен приводить к одному и тому же *конечному состоянию* ресурса инфраструктуры. В конце концов, когда я пишу автоматизацию, у меня есть одна главная цель: возможность запускать автоматизацию несколько раз и получать один и тот же результат.

Давайте рассмотрим пример, поясняющий необходимость в более строгих требованиях. Представьте, что вы написали сценарий, который, кроме прочих действий, настраивает сетевой интерфейс, а затем выполняет перезагрузку. При первом запуске сценария происходят настройка сетевого коммутатора и его перезагрузка. Вы сохраняете этот сценарий как версию 1. Через несколько месяцев ваш коллега по команде просит вас снова запустить сценарий на коммутаторе.

Вы запускаете сценарий, и коммутатор перезагружается. Однако перезагрузка отключает некоторые критически важные приложения! Вы уже настроили сетевой интерфейс раньше. Зачем нужна перезагрузка коммутатора?

Что ж, есть способ предотвратить перезагрузку коммутатора, если сетевой интерфейс уже настроен. На рис. 1.8 показано, как вы соз-

даете версию 2 сценария и добавляете условный оператор `if`. Оператор проверяет, настроен ли интерфейс перед перезагрузкой коммутатора. При повторном запуске скрипта версии 2 приложения не отключаются.

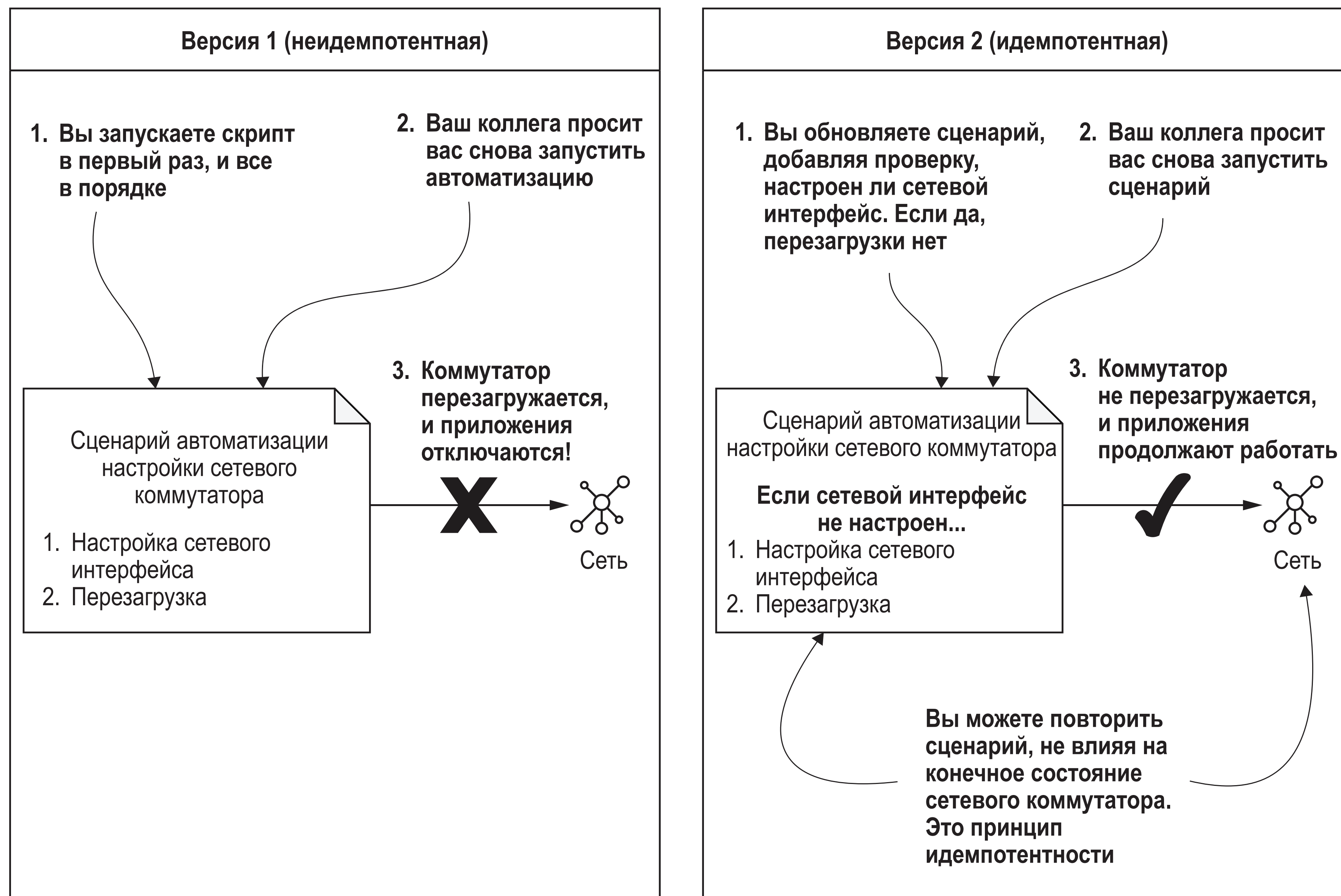


Рис. 1.8 В версии 1 сценария коммутатор перезагружается при каждом запуске сценария. В версии 2 перед перезагрузкой коммутатора вы проверяете, настроен ли сетевой интерфейс. Это сохраняет рабочее состояние сети

Наличие этой проверки соответствует принципу идемпотентности. *Идемпотентность* (idempotency) гарантирует, что вы сможете повторно запустить сценарий автоматизации, не затрагивая инфраструктуру, если только вы не измените конфигурацию или не произойдет дрейф. Если конфигурация инфраструктуры или скрипт идемпотентны, вы можете повторно запустить автоматизацию несколько раз, не влияя на состояние или работоспособность ресурса.

ОПРЕДЕЛЕНИЕ Принцип идемпотентности гарантирует, что вы можете многократно запускать сценарий автоматизации в инфраструктуре, не влияя на ее конечное состояние и не вызывая каких-либо побочных эффектов. Вы должны влиять на инфраструктуру только тогда, когда обновляете ее атрибуты автоматизации.

Почему вам следует соблюдать принцип идемпотентности в IaC, например в случае вашего сетевого сценария? В этом примере не-

обходимо избежать перезагрузки сетевого коммутатора, чтобы сеть работала без перебоев. Вы уже настроили сетевой интерфейс; зачем настраивать его еще раз? Вам нужно будет настроить интерфейс, только если он не существует или требует изменений.

Без идемпотентности ваш сценарий автоматизации может случайно испортить инфраструктуру. Например, вы можете повторить сценарий и создать новый набор серверов, удвоив их количество. Еще катастрофичнее ситуация, когда сценарий автоматического обновления БД удаляет критическую базу данных! Для соблюдения принципа идемпотентности необходимо добавить в сценарий проверку текущего состояния инфраструктуры. Обычно перед запуском автоматизации следует включать в сценарий несколько условных операторов, проверяющих, соответствует ли конфигурация ожидаемой. Условные операторы помогают применять изменения, когда это необходимо, и избегать побочных эффектов, которые могут повлиять на работоспособность инфраструктуры.

Разработка идемпотентных сценариев снижает риск, поскольку поощряет применение логики для сохранения конечного ожидаемого состояния системы. Если автоматизация будет вызывать фатальные сбои в инфраструктуре, организации не захотят использовать IaC из-за потенциального риска. Когда вы узнаете, как безопасно выполнять развертывание изменений и предварительно просматривать изменения автоматизации перед развертыванием (в главе 11), идемпотентность станет ключевым принципом вашей деятельности.

1.3.3 *Компонуемость*

Для нормальной работы вам нужна возможность составлять любой набор компонентов инфраструктуры, независимо от инструмента или конфигурации. Вам также необходимо обновлять отдельные конфигурации, не затрагивая всю систему. Оба этих требования привели к появлению подходов модульности и разделения инфраструктурных зависимостей, о которых вы узнаете больше в главах 3 и 4.

Например, представьте, что вы создаете инфраструктуру для приложения, к которому кто-то получает доступ на сайте hello-world.com. Вот минимальный перечень ресурсов, необходимых для создания безопасной и готовой к работе конфигурации:

- сервер;
- балансировщик нагрузки;
- частная сеть для сервера;
- публичная сеть для балансировщика нагрузки;
- правило маршрутизации, разрешающее трафик из частной сети;
- правило маршрутизации, разрешающее внешний трафик к балансировщику нагрузки;

- правило маршрутизации, разрешающее трафик от балансировщика нагрузки на сервер;
- доменное имя `hello-world.com`.

Вы можете написать эту конфигурацию с нуля. Но что, если вы нашли готовые *модули*, т. е. группы компонентов инфраструктуры, которые можно использовать для сборки системы? Теперь у вас есть несколько модулей, которые содержат следующие компоненты:

- сети (частные и общедоступные сети, шлюзы для маршрутизации трафика из частной сети, правила маршрутизации, разрешающие трафик из частной сети);
- серверы;
- балансировщики нагрузки (доменное имя, правила маршрутизации, разрешающие трафик от балансировщика нагрузки на сервер).

На рис. 1.9 показано, как вы выбираете модули сети, сервера и балансировщики нагрузки для создания промышленной среды. Позже вы приходите к выводу, что вам нужен балансировщик нагрузки премиум-класса. Вы заменяете стандартный балансировщик нагрузки на премиальный, чтобы иметь возможность обслуживать больше трафика. Сервер и сеть продолжают работать, не затрагивая пользователей.

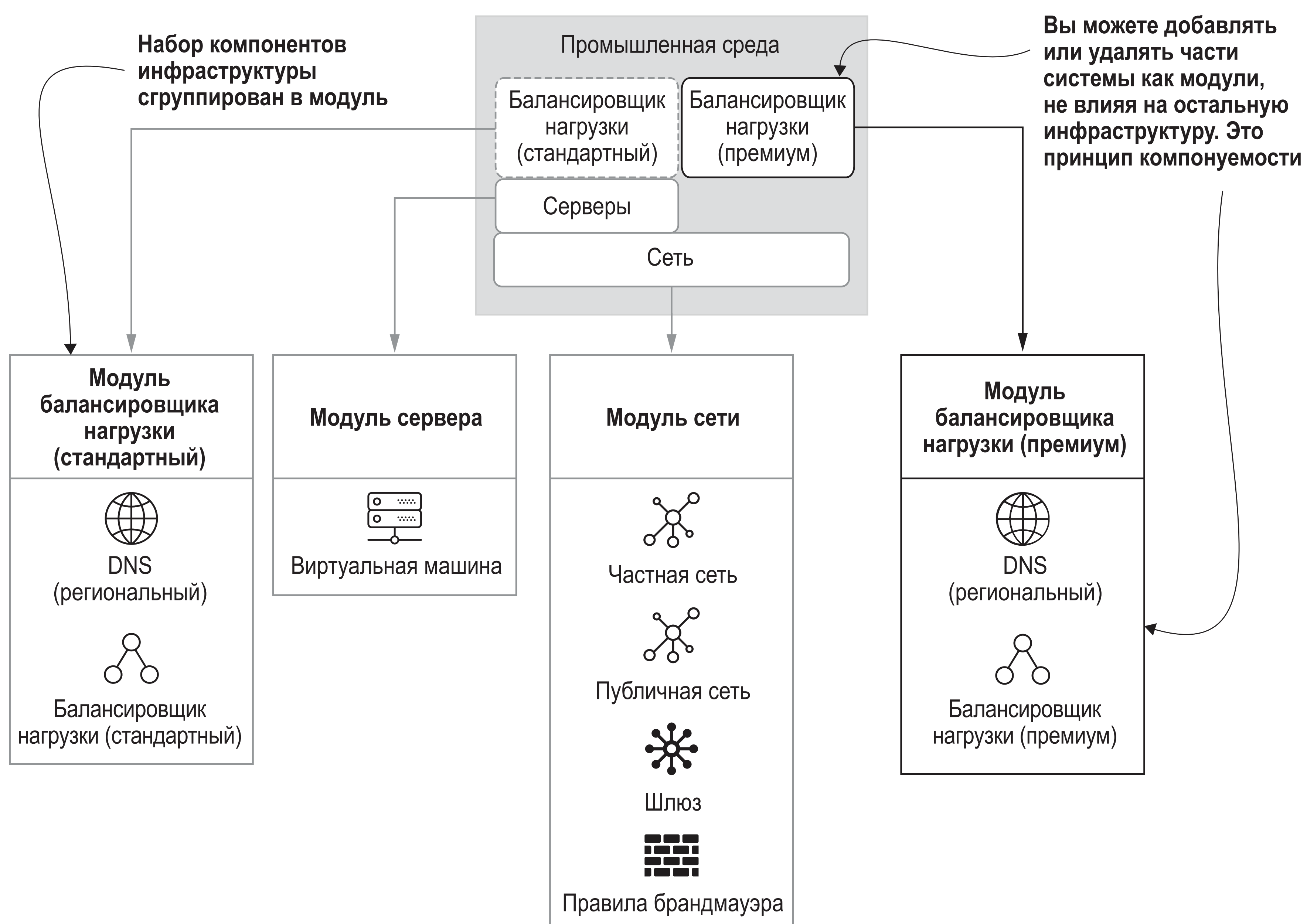


Рис. 1.9 Вы создаете свою промышленную среду из типовых компонентов инфраструктуры, чтобы можно было легко добавлять новые ресурсы, например балансировщик нагрузки премиум-класса

Ваш коллега по команде может даже добавить в среду базу данных, не затрагивая балансировщик нагрузки, серверы или сеть. Вы группируете и выбираете ресурсы инфраструктуры в различных комбинациях, что соответствует принципу компонентности.

ОПРЕДЕЛЕНИЕ *Принцип компонентности* означает, что вы можете собрать любую комбинацию ресурсов инфраструктуры и обновлять каждый из них по отдельности, не затрагивая другие.

Чем выше компонентность вашей конфигурации, тем проще создавать новые системы с меньшими усилиями. Подумайте о построении вашей IaC из стандартных модулей. Наверняка вы захотите иметь возможность обновлять или развивать подмножества ресурсов, не подвергая риску сбоя всю систему! Если вы не придерживаетесь принципа компонентности IaC, вы рискуете сбоями при внесении изменений из-за неизвестных зависимостей в сложных инфраструктурных системах.

Применение компонентности облегчает масштабирование инфраструктуры организации и дает командам возможность безопасно взаимодействовать с инфраструктурными системами. В главах 3 и 4 будут рассмотрены некоторые типовые приемы, которые помогут вам повысить модульность инфраструктуры и улучшить компонентность.

1.3.4 Развиваемость

Рано или поздно вам придется учитывать масштаб и рост вашей системы, но не хотелось бы оптимизировать конфигурацию слишком рано и без необходимости. Большая часть конфигурации инфраструктуры со временем изменится, включая ее архитектуру.

Допустим, сначала вы дали ресурсу инфраструктуры имя `example`. Позже вам нужно будет изменить имя ресурса на `production`. Вам предстоит выполнить поиск и замену сотен тегов, имен, зависимых ресурсов и многого другого. Процедура поиска и замены требует больших усилий.

Вы заметили, что забыли исправить некоторые поля, и внесение изменений в инфраструктуру фактически не удалось. Чтобы избежать проблем при будущем изменении имен, тегов и других метаданных, вы создаете переменные для имен, и конфигурация ссылается на эти переменные. На рис. 1.10 показано, как вы обновляете глобальную переменную `NAME`, и изменение распространяется на всю систему.

Этот пример выглядит слишком простым. Почему мы уделяем столько внимания смене имени? Конфигурация IaC, построенная с учетом принципа развиваемости, сводит к минимуму усилия (время и затраты) на изменение системы и риск неудачи при изменении.

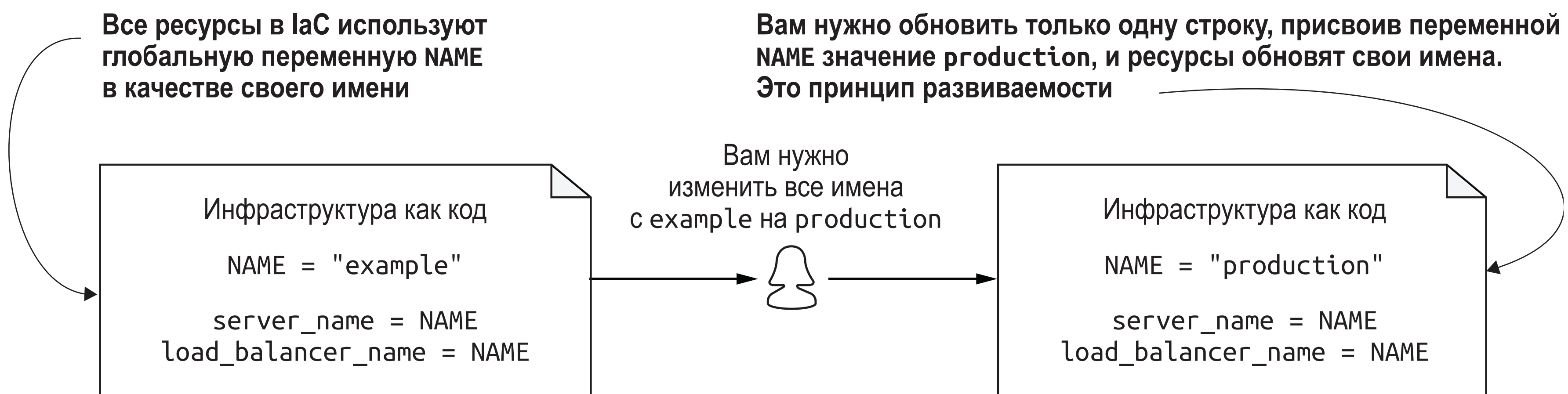


Рис. 1.10 Вместо того чтобы искать и заменять все экземпляры имени, вы можете установить переменную верхнего уровня с именем для всех ресурсов

ОПРЕДЕЛЕНИЕ *Принцип развиваемости* гарантирует, что вы можете изменять ресурсы инфраструктуры в соответствии с масштабированием или ростом системы, сводя при этом к минимуму трудозатраты и риск сбоя.

Развитие системы включает в себя не только незначительные изменения, такие как изменение имени. Более радикальное изменение в архитектуре инфраструктуры может, например, привести к замене Google Cloud Bigtable на Amazon Elastic Map Reduce (EMR). В этом случае безболезненный ввод нового приложения в эксплуатацию можно выполнить с помощью Apache HBase – распределенной базы данных с открытым исходным кодом, которая поддерживает оба варианта и просто требует замены конечной точки базы данных.

Мы учитываем это развитие системы в IaC, указывая конечную точку базы данных, к которой обращается приложение, и незаметно завершаем обновление за счет наличия готовой конфигурации для обоих вариантов. После тестирования базы данных Amazon Web Services (AWS) мы просто указываем ее конечную точку для использования приложением.

ПРИМЕЧАНИЕ В этой книге я не полностью раскрываю теорию развития вашей архитектуры. Я настоятельно рекомендую книгу *Building Evolutionary Architectures* (Neal Ford et al., O'Reilly, 2017), если вы хотите узнать больше. В книге обсуждается построение инфраструктуры с учетом развиваемости.

Есть риск, что вам будет очень трудно развивать свою систему, потому что вы не использовали паттерны и приемы, помогающие ей меняться. Хорошая конфигурация IaC опирается на методы, способствующие будущему развитию. Многие главы этой книги содержат описания паттернов, облегчающих развитие инфраструктуры и сводящих к минимуму влияние изменений на критически важные системы.

1.3.5 *Применение принципов разработки*

Определения воспроизводимости, идемпотентности, компоуемости и развиваемости выглядят разрозненными. Но вместе они помогают ограничить архитектуру вашей инфраструктуры и определяют поведение многих инструментов IaC. Разработанное вами решение IaC должно соответствовать всем четырем принципам. Только так вы сможете обеспечить масштабируемость решений, эффективную совместную работу и рост компании. На рис. 1.11 кратко суммированы эти четыре важных принципа и их определения.

Воспроизводимость	Идемпотентность	Компоуемость	Развиваемость
Можно ли использовать конфигурацию для воспроизведения среды или ресурсов инфраструктуры? Среда разработки Окружение Конфигурация	Можно ли многократно запускать сценарий в инфраструктуре, не меняя ее текущего состояния? Я выполняю... ☐ шаг 1 ☐ шаг 2 ☐ шаг 3 Они дают одинаковое конечное состояние Если я выполнил шаг 1, я выполняю... ☐ шаг 2 ☐ шаг 3	Можно ли собрать любую комбинацию ресурсов инфраструктуры, не создавая новую систему? Вы можете легко добавить кеш в систему Оркестратор контейнеров Виртуальная машина Сеть Кеш	Можно ли изменить инфраструктуру и свести к минимуму нарушение работы всей системы? Вы можете заменить базу данных и свести к минимуму нарушение работы своих приложений Приложения Реляционная база данных Сеть БД NoSQL

Рис. 1.11 Ваше решение IaC должно быть воспроизводимым, идемпотентным, компоуемым и развиваемым. Задайте себе несколько вопросов, чтобы определить, соответствует ли оно всем четырем принципам

В процессе разработки IaC постоянно следите за соответствием всем четырем принципам. Они помогут вам писать и публиковать конфигурацию IaC с меньшими усилиями и в идеале минимизировать негативное влияние изменений на вашу систему. Несоблюдение какого-либо принципа может препятствовать обновлению ресурсов инфраструктуры или увеличить потенциальный ущерб.

Следует также задуматься о том, соответствуют ли ваши конфигурации или инструменты практическим потребностям. Например, задайте следующие вопросы о вашем инструменте:

- позволяет ли этот инструмент воссоздавать целые среды?
- что произойдет, если вы перезапустите инструмент для принудительной настройки?

- можете ли вы смешивать и сочетать различные фрагменты конфигурации, чтобы создать новый набор компонентов инфраструктуры?
- обладает ли инструмент возможностью развивать ресурс инфраструктуры, не влияя на другие системы?

В этой книге я придерживаюсь принципов, позволяющих ответить на все упомянутые вопросы, и стараюсь дать вам навыки тестирования, обновления и развертывания инфраструктуры с учетом отказоустойчивости и масштабируемости.

Упражнение 1.1

Выберите скрипт или конфигурацию инфраструктуры в своей организации. Оцените их соответствие принципам IaC. Способствуют ли они воспроизводимости и идемпотентности, помогают ли в компоновке и облегчают ли развитие?

Ответы на упражнения даны в приложении В.

1.4 Кто и зачем использует IaC?

IaC обычно относят к методикам DevOps. Однако для использования IaC не обязательно применять DevOps во всей организации. Если вдуматься, вы всего лишь стремитесь управлять своей инфраструктурой таким образом, чтобы снизить частоту неудачных изменений и среднее время устранения сбоя (mean time to resolution, MTTR), чтобы спокойно спать по выходным в качестве оператора или тратить больше времени на написание кода в качестве разработчика. Есть несколько причин использовать IaC, даже если вы считаете, что этот подход вам не нужен.

1.4.1 Управление изменениями

Представьте, что вы применили изменение к определенной инфраструктуре и обнаружили, что оно стало причиной серьезного сбоя. Организации пытаются предотвратить сбои с помощью методики *управления изменениями* (change management) – набора шагов и проверок, которые гарантируют, что ваши изменения не навредят производству. Процесс управления изменениями часто включает в себя *панель обзора изменений* (change review board) для просмотра изменений или назначение *окна изменений* (change window), где запланировано безопасное время для проведения работ.

ОПРЕДЕЛЕНИЕ *Управление изменениями* – это методика, описывающая набор шагов и проверок, которые компания

предпринимает для внедрения изменений в промышленную среду и предотвращения связанных с этим сбоев.

Тем не менее ни одно изменение не обходится без риска. Применение практик IaC помогает снизить риск изменений за счет модульности инфраструктуры (глава 3) и безопасного развертывания изменений (глава 11), ограничивая тем самым объем потенциального ущерба.

В одном прискорбном случае я поленилась использовать IaC для снижения риска изменений. Мне понадобилось развернуть на серверах новый бинарный файл, после чего требовался перезапуск некоторых зависимых служб. Я написала сценарий, попросила коллег проверить его, и комиссия по анализу изменений разрешила его запустить. Изменения применили на выходных, а в понедельник я получила несколько сообщений о том, что серверы, поддерживающие приложение для синхронизации данных, моментально вышли из строя. Мой товарищ по команде объяснил этот сбой несовместимостью старой операционной системы серверов с зависимостями в моем сценарии.

Оглядываясь назад, я понимаю, что соблюдение принципов IaC могло бы снизить риск сбоя. Когда я задним числом применила к изменениям подход RICE, то поняла, что проигнорировала следующие принципы:

- воспроизводимость – я не воспроизводила свой сценарий на различных тестовых экземплярах, имитируя различные серверы;
- идемпотентность – я не предусмотрела логику для проверки версии операционной системы перед запуском обновления;
- компонуемость – я не ограничивала радиус действия изменения небольшим набором менее важных серверов;
- развиваемость – я не обновляла операционную систему серверов и не уменьшала вариативность инфраструктуры.

Уменьшение вариативности позволяет развивать инфраструктуру и снижать риски, поскольку фактическая конфигурация соответствует ожидаемой в ходе автоматизации, что делает применение изменений более простым и надежным. В главе 7 мы обсудим, как внедрить IaC в ваш процесс управления изменениями.

1.4.2 *Окупаемость затрат времени*

Внедрение IaC в первую очередь требует затрат времени, и необходимость в этих затратах бывает сложно объяснить руководству, особенно если ваши устройства или оборудование не имеют подходящих интерфейсов автоматизации. В самом деле, сложно оправдать трату рабочего времени на автоматизацию задачи, которую вы выполняете только раз в год или даже десятилетие, особенно если эту

автоматизацию нужно внедрить с нуля. Хотя реализация IaC требует затрат рабочего времени, в долгосрочной перспективе она окупает себя за счет сокращения времени на внесение изменений. Как именно это работает?

Представьте, что вам нужно обновить один и тот же пакет на 10 серверах. Раньше вы делали это без IaC. Вы вручную входили в систему, обновляли пакет, проверяли, что все работает, исправляли ошибки и переходили к следующему серверу. В среднем вы тратили на обновление 10 ч.

На рис. 1.12 показано, что изменения, вносимые без IaC, требуют постоянных затрат времени. Если у вас есть дополнительные изменения, вы можете потратить еще несколько часов на исправление или обновление системы. Неудачное изменение может означать, что вы потратите несколько дней на устранение сбоя.

Допустим, вы решили потратить время на создание IaC для этих серверов (сплошная линия на рис. 1.12). Вы ставите перед собой задачу ускорить обновление конфигурации серверов, которое занимает около 40 ч. Вы один раз выделяете время на разработку IaC, а потом тратите менее 5 минут на обновление всех серверов каждый раз, когда вносите изменения.

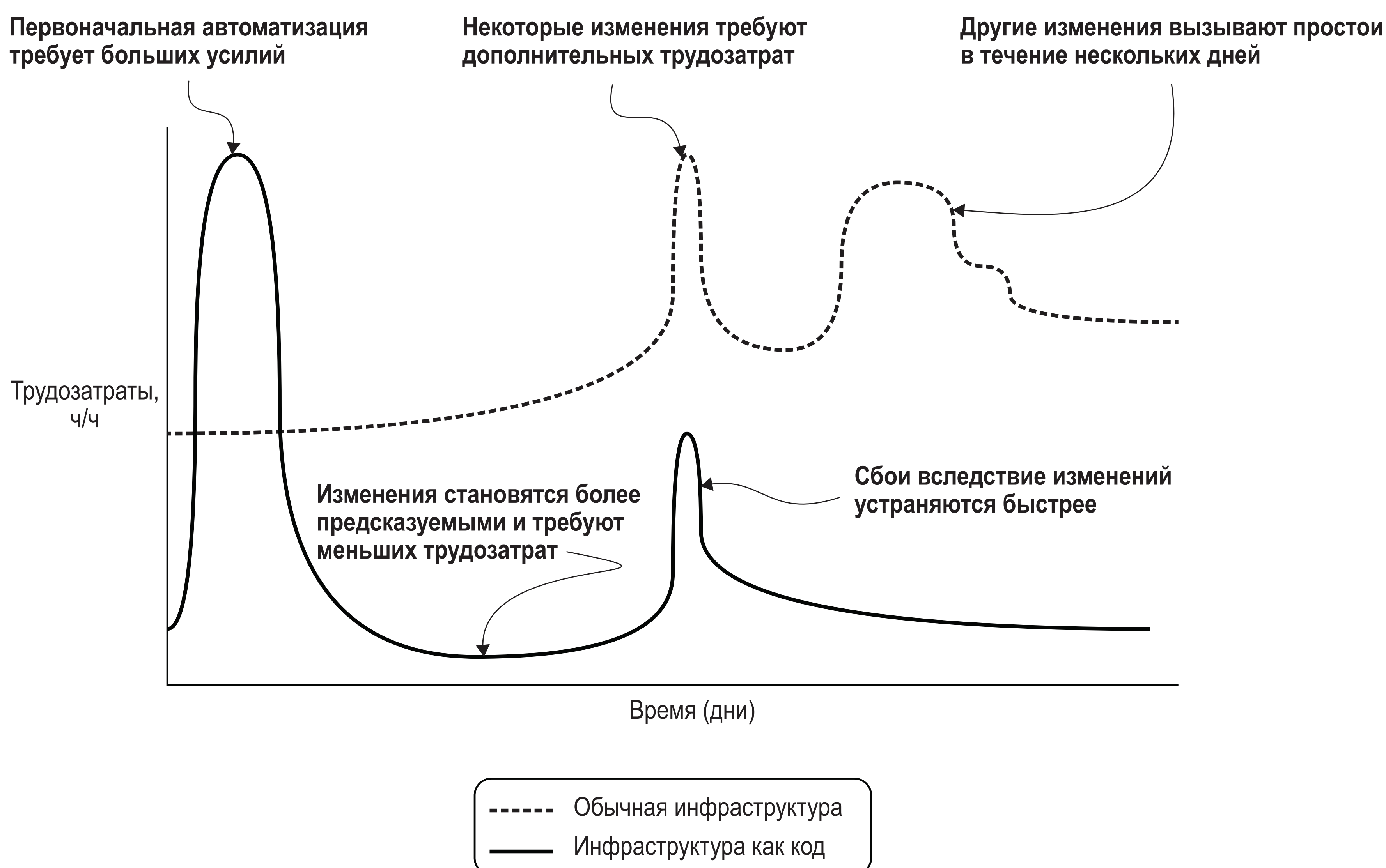


Рис. 1.12 После больших первоначальных усилий по автоматизации трудозатраты значительно снижаются. Без IaC время внесения изменений может сильно отклоняться в худшую сторону

Затраты времени на внедрение IaC напрямую связаны с трудозатратами. Профилактика значительно сокращает усилия по исправ-

лению последствий. Без IaC вы можете потратить недели, пытаясь устранить серьезный сбой в системе. Обычно вы тратите эти недели, пытаясь перекроить внесенные вручную изменения, отменить определенные изменения или, в худшем случае, построить новую систему с нуля.

Вы должны потратить время и силы на разработку IaC, даже если предстоящая работа вас пугает. Это инвестиция в себя, поскольку вы потратите меньше времени на отладку неисправных конфигураций или восстановление неисправных систем в долгосрочной перспективе. Если однажды ваша система полностью выйдет из строя, вы можете легко воспроизвести ее, запустив сценарий IaC.

Автоматизация и тестирование способствуют предсказуемости и снижают ущерб в случае сбоя. Они снижают частоту неудачных изменений и среднее время восстановления отказавших систем. По мере развития и масштабирования вашей инфраструктуры вы сможете использовать подробные методы тестирования, описанные в этой книге, чтобы снизить долю неудачных обновлений и уменьшить трудозатраты на подготовку будущих изменений.

1.4.3 Обмен знаниями

Сценарий IaC отражает архитектуру и конфигурацию инфраструктуры, что способствует снижению количества человеческих ошибок и повышает надежность. Однажды инженер сказал мне: «Нам не нужен сценарий IaC для сетевого коммутатора третичного пассивного центра обработки данных (используемого для резервного копирования). В любом случае, это буквально пара настроек, которые я внесу один раз, и больше незачем к нему прикасаться».

Инженер покинул организацию вскоре после настройки коммутатора. Позже моей команде потребовалось преобразовать третичный пассивный центр обработки данных в активный в соответствии с новыми потребностями компании. Нам пришлось в суете и панике делать обратный инжиниринг чужой конфигурации коммутатора. Мы потратили почти два месяца, чтобы разобраться с сетевым подключением, переработать его конфигурацию и добавить управление с помощью IaC.

Даже если задача не имеет четкой формулировки или команда, настраивающая инфраструктуру, состоит из одного человека, затраты времени и усилий на изучение и внедрение IaC значительно облегчат вашу жизнь, особенно по мере роста инфраструктурных систем и численности команд. Со временем вы обнаружите, что автоматизация занимает намного меньше времени, чем попытки запомнить чужую конфигурацию сервера или рассказать о ней новому члену команды.

Описание задачи в виде кода (этот подход также называют «код как документация») передает ожидаемое состояние инфраструктуры и архитектуры системы.

ОПРЕДЕЛЕНИЕ Код как документация означает, что код передает предназначение программного обеспечения или системы без потребности в дополнительной справочной документации.

Человек, незнакомый с системой, должен иметь возможность изучить конфигурацию инфраструктуры и понять ее назначение, имея доступ только к коду. Наш мир не идеален, и мы не можем ожидать, что код будет служить исчерпывающей документацией. Однако код должен отражать большую часть вашей инфраструктурной архитектуры и ожиданий от системы.

1.4.4 Безопасность

Аудит и проверка конфигураций в IaC помогают выявить проблемы безопасности на более ранних этапах процесса разработки. Иногда это называют *переносом безопасности влево* (shifting security left). Чем раньше вы начнете проверки безопасности в своем рабочем процессе, тем меньше будет риск столкнуться с уязвимостями и сбоями системы в промышленной среде. О методах обеспечения безопасности более подробно говорится в главе 8.

Например, вы можете временно расширить доступ к объектному хранилищу, чтобы все могли выполнять чтение и запись в процессе разработки. Затем вы развертываете конфигурацию в промышленной среде, но забываете вернуть ограничения. В результате некоторые объекты в хранилище остались общедоступными. Хотя это кажется простой ошибкой, она может иметь очень серьезные последствия, если объект содержит данные о покупателях интернет-магазина.

ПРИМЕЧАНИЕ Вы легко найдете в новостях многочисленные примеры инцидентов с неправильно сконфигурированными объектными хранилищами, из-за чего была раскрыта информация о водительских правах, или даже случаев утечек баз данных с паролем по умолчанию, в которых хранились личные данные миллионов владельцев кредитных карт. Некоторые нарушения безопасности связаны с уязвимостями сторонних решений, но у многих причиной является небезопасная конфигурация. Организации, которые предотвращают небезопасные настройки на ранних этапах, обычно могут быстро изучить конфигурацию, провести аудит контроля доступа, оценить объем ущерба и устранить нарушение.

IaC упрощает управление доступом за счет наличия единой конфигурации. С помощью IaC вы можете протестировать конфигурацию, чтобы убедиться, что объектное хранилище не разрешает публичный доступ. Кроме того, вы можете добавить проверку пос-

ле развертывания в промышленной среде, чтобы еще раз удостовериться, что ваша политика разрешает доступ только к нужным объектам. Даже политики безопасности в центре обработки данных, такие как правила брандмауэра, можно описать в сценарии IaC, а затем убедиться, что они разрешают только входящие соединения из известных источников.

Если вы столкнулись с нарушением безопасности, IaC поможет протестировать конфигурацию, быстро проверить контроль доступа, оценить объем ущерба и устранить нарушение. Вы можете использовать одни и те же методы IaC для внесения различных изменений. В этой книге вы познакомитесь с некоторыми практическими приемами аудита безопасности и защиты инфраструктуры в соответствии с принципами IaC.

1.5 Инструменты

Инструментарий IaC довольно сильно разнится, поскольку применяется к различным ресурсам. Большинство инструментов предназначены для одного из трех вариантов использования, включая следующие:

- инициализация инфраструктуры;
- управление конфигурацией;
- создание образа.

В этой книге я уделяю основное внимание инструментам, применяемым для инициализации инфраструктуры, которые развертывают наборы инфраструктурных ресурсов и управляют ими. Я добавила в книгу несколько примечаний и примеров по управлению конфигурацией и созданию образов, чтобы подчеркнуть различия в подходах.

1.5.1 Примеры в этой книге

В этой книге я столкнулась с проблемой создания наглядных примеров, не зависящих от инструментов или платформ. Мне как автору книги о паттернах и методах нужно было найти способ выразить концепции на общем языке программирования, не переписывая логику инструмента.

Python и Terraform

На рис. 1.13 показан рабочий процесс с листингами кода и примерами. Техническая реализация подробно описана в приложении А. Я написала листинги кода на Python, чтобы создать файл JavaScript Object Notation (JSON), поступающий затем в Terraform – инструмент инициализации инфраструктуры, поддерживающий интеграцию с публичными облаками и другими поставщиками услуг.

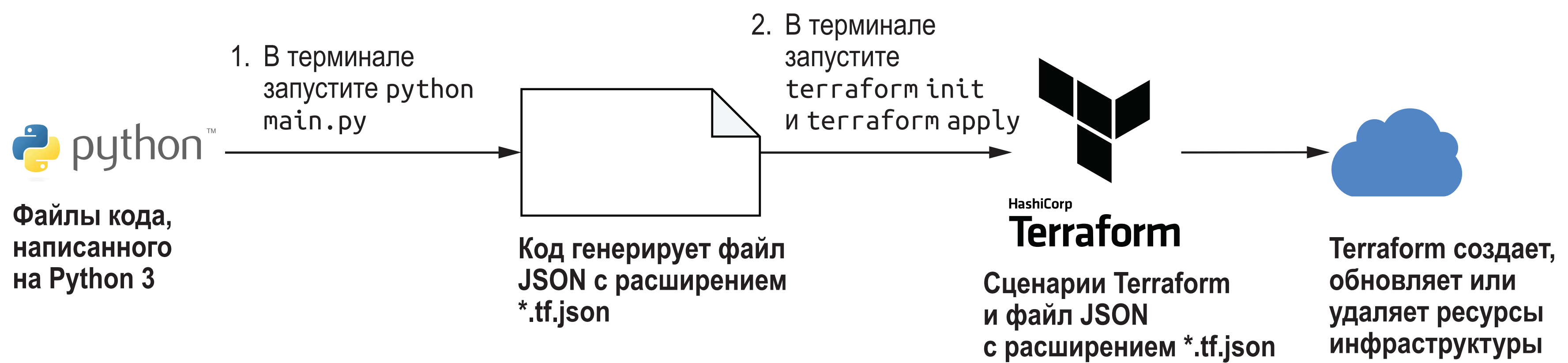


Рис. 1.13 В примерах этой книги код Python создает файл JSON, который затем поступает в Terraform

Когда вы запускаете скрипт Python с помощью команды `python main.py`, код создает файл JSON с расширением `*.tf.json`. В файле JSON используется синтаксис, специфичный для Terraform. Затем вы можете перейти в каталог с файлом `*.tf.json` и выполнить `terraform init` и `terraform apply` для создания ресурсов. Хотя может показаться, что код Python добавляет ненужную абстракцию, он гарантирует, что я могу предложить конкретные примеры, не зависящие от платформы и инструмента.

Я понимаю, что такой рабочий процесс выглядит необоснованно сложным. Однако эта сложность служит двум целям. Файлы Python предоставляют обобщенную реализацию паттернов и методов на языке программирования. Конфигурация JSON позволяет вам запускать и создавать ресурсы с помощью инструмента, вместо того чтобы самому создавать абстракции.

ПРИМЕЧАНИЕ Полные примеры кода можно найти в репозитории кода этой книги на GitHub: <https://github.com/joatmon08/manning-book>.

Вам не обязательно хорошо знать Python или Terraform, чтобы понять примеры кода. Если вы хотите запустить примеры и создать ресурсы, достаточно прочитать вводное руководство по Terraform или Python, посвященное синтаксису и командам.

ПРИМЕЧАНИЕ Желающим ознакомиться с дополнительными ресурсами по Terraform и Python я рекомендую книги *Terraform in Action* (Scott Winkler, Manning, 2021), *The Quick Python Book* (Naomi R. Cedar, Manning, 2018) или *Python Workout* (Reuven M. Lerner, Manning, 2020).

Google Cloud Platform

Хотя в других публикациях чаще встречаются примеры для AWS или Microsoft Azure, я решила использовать Google Cloud Platform (GCP) в качестве основного поставщика облачных услуг по трем основным причинам. Во-первых, GCP требует в целом меньше ресурсов

для получения аналогичной архитектуры. Это сокращает примеры и позволяет сосредоточиться на закономерностях и методах, а не на конфигурации.

Во-вторых, в услугах GCP используются более простые имена ресурсов и общая инфраструктурная терминология. Если вы работаете в центре обработки данных, вы все равно без труда поймете, что создают сервисы. Например, Google Cloud SQL создает базы данных SQL.

Если вы запустите примеры, то будете использовать следующие ресурсы GCP:

- сеть (сети, балансировщики нагрузки и межсетевые экраны);
- вычислительные ресурсы;
- управляемые очереди (Pub/Sub);
- хранилище (Cloud Storage);
- управление идентификацией и доступом (IAM);
- службы Kubernetes (Kubernetes Engine и Cloud Run);
- базы данных (Cloud SQL).

Вам *не* обязательно знать технические детали каждой услуги. Я использую их, чтобы продемонстрировать управление зависимостями между ресурсами инфраструктуры. Я старалась не использовать специализированные сервисы, которые предназначены для конкретной облачной платформы, например машинного обучения.

Эквиваленты AWS и Azure

Каждый пример сопровождается примечаниями с описанием эквивалентных сервисов AWS и Azure для дальнейшего закрепления конкретных паттернов и методов. Чтобы перенести примеры на платформу выбранного вами поставщика облачных услуг, вам может потребоваться внести изменения в код и исправить некоторые зависимости. Например, вы можете создавать сети GCP со встроенными шлюзами по умолчанию, тогда как в сетях AWS эти шлюзы надо определить в явном виде.

Некоторые примеры имеют эквивалент AWS в репозитории кода книги (<https://github.com/joatmon08/manning-book>). Дополнительную информацию о настройке AWS или Azure для работы с примерами вы также найдете в приложении А.

Третья причина, по которой я выбрала GCP, связана с затратами. В GCP есть условно бесплатный тарифный план. Если вы создадите новую учетную запись в GCP, вы получите доступ к пробной версии с бонусом на сумму до 300 долл. США (на момент написания книги). Почти все примеры можно выполнить бесплатно, если у вас есть подходящая учетная запись. Я отдельно отмечаю все ресурсы, которые не соответствовали бесплатному плану на момент написания этой книги.

Использование Google Cloud Platform

Для получения дополнительной информации о бесплатной программе GCP посетите ее веб-страницу по адресу <https://cloud.google.com/free>.

Я рекомендую создать отдельный проект GCP для запуска всех примеров. Отдельный проект позволяет изолировать ваши ресурсы. Когда вы закончите читать эту книгу, вы сможете удалить проект и его ресурсы. Ознакомьтесь с руководством по созданию проектов GCP по адресу <http://mng.bz/e7QG>.

1.5.2 Инициализация инфраструктуры (provisioning)

Инструменты инициализации или начальной настройки (provisioning tools) предназначены для создания и управления наборами инфраструктурных ресурсов конкретного провайдера, будь то общедоступное облако, центр обработки данных или решение для мониторинга на хостинге. Под *провайдером* (provider) здесь понимается центр обработки данных, IaaS, PaaS или SaaS, ответственные за предоставление ресурсов инфраструктуры.

ОПРЕДЕЛЕНИЕ Инструмент инициализации отвечает за создание и управление набором ресурсов инфраструктуры для общедоступного облака, центра обработки данных или хостинга.

Некоторые инструменты инициализации работают только с конкретным провайдером, тогда как другие интегрируются с несколькими провайдерами (табл. 1.1).

Таблица 1.1 Примеры инструментов инициализации и провайдеров

Инструмент	Провайдер
AWS CloudFormation	Amazon Web Services
Google Cloud Deployment Manager	Google Cloud Platform
Azure Resource Manager	Microsoft Azure
Bicep	Microsoft Azure
HashiCorp Terraform	Разные (полный список на www.terraform.io/docs/providers/index.html)
Pulumi	Разные (полный список на www.pulumi.com/docs/intro/cloud-providers/)
AWS Cloud Development Kit	Amazon Web Services
Kubernetes manifests	Kubernetes (оркестратор контейнеров)

Большинство инструментов инициализации могут просматривать изменения в системе и выражать зависимости между ресурсами инфраструктуры, что также называется *пробным прогоном* (dry run).

ОПРЕДЕЛЕНИЕ *Пробный прогон* анализирует и отображает ожидаемые изменения в инфраструктуре, прежде чем вы примените эти изменения к ресурсам.

Например, вы можете указать зависимость между сетью и сервером. Если вы измените сеть, инструмент инициализации покажет, что сервер также может измениться.

1.5.3 Управление конфигурацией

Инструменты управления конфигурацией (configuration management tool) предназначены для приведения серверов и компьютерных систем к желаемому состоянию (конфигурации). Большинство инструментов управления конфигурацией превосходно подходят для начальной и текущей настройки устройств, например при развертывании и обслуживании сервера.

ОПРЕДЕЛЕНИЕ *Инструмент управления конфигурацией* настраивает набор серверов, т. е. определяет ресурсы, пакеты и компоненты, размещенные на них.

Например, если в вашем центре обработки данных имеется 10 000 серверов, как вы можете быть уверены, что абсолютно на всех из них работает определенная версия пакетов, одобренных вашей командой безопасности? Вы не сможете регулярно заходить на 10 000 серверов и вручную вводить команды для проверки версии. Если вы настраиваете серверы с помощью инструмента управления конфигурацией, вы можете запустить одну команду, чтобы просмотреть все 10 000 серверов и выполнить обновления пакетов.

Вот неполный список инструментов управления конфигурацией, которые хорошо решают проблемы в этой области:

- Chef;
- Puppet;
- Red Hat Ansible;
- Salt;
- CFEngine.

Хотя эта книга посвящена инструментам инициализации и управлению системами с несколькими поставщиками, я рассмотрю некоторые приемы управления конфигурацией, связанные с тестированием инфраструктуры, ее обновлением и безопасностью. Инструменты управления конфигурацией помогут вам настроить сервер и сетевую инфраструктуру.

ПРИМЕЧАНИЕ Для получения дополнительной информации я рекомендую приобрести книги и учебные пособия по выбранному вами инструменту управления конфигурацией.

В них вы найдете более подробное руководство, соответствующее их подходу к проектированию.

В дополнение к уже имеющейся путанице некоторые инструменты управления конфигурацией предлагают интеграцию с центрами обработки данных и провайдерами облачных услуг. В результате у вас может возникнуть идея использовать ваш инструмент конфигурации в качестве инструмента инициализации. Хотя это возможно, такое решение не всегда оптимально, поскольку инструменты инициализации инфраструктуры часто применяют другой подход к проектированию для устранения зависимостей между ресурсами. Я рассмотрю этот нюанс подробнее в следующей главе.

1.5.4 Создание образа

При создании сервера необходимо указать образ машины с операционной системой. *Инструменты создания образов* создают образы, применяемые во время выполнения приложений, будь то контейнер или сервер.

ОПРЕДЕЛЕНИЕ *Инструмент создания образов* создает образы машин для среды выполнения приложений, таких как контейнеры или серверы.

Большинство генераторов образов позволяют указать среду выполнения и цели сборки. В табл. 1.2 описаны несколько инструментов, поддерживаемые ими среды выполнения, а также цели сборки и платформы.

Таблица 1.2 Примеры генераторов образов и их связь с поставщиками

Инструмент	Среда выполнения	Цель сборки
HashiCorp Packer	Контейнеры и серверы	Разные (полный список на сайте www.packer.io/docs/builders)
Docker	Контейнеры	Регистры контейнеров
EC2 Image Builder	Серверы	Amazon Web Services
Azure VM Image Builder	Серверы	Microsoft Azure

В этой книге я не рассматриваю вопросы, связанные с созданием образов. Однако паттерны тестирования, поставки и обеспечения соответствия в главах 6–8 снабжены комментариями относительно создания образов. В следующей главе вы узнаете о неизменности – важнейшей парадигме, от которой зависит работа инструмента создания образов.

На рис. 1.14 показано, как инструменты создания образов, управления конфигурацией и инициализации работают вместе. Процесс развертывания новой конфигурации сервера часто начинается

с инструмента управления конфигурацией, когда вы закладываете основу инфраструктуры и проверяете правильность конфигурации вашего сервера.

После настройки нужной конфигурации сервера вы используете инструмент создания образов, чтобы сохранить образ сервера с его версиями и средой выполнения. Наконец, ваш инструмент инициализации использует снимок инструмента создания образов для создания новых рабочих серверов с нужной вам конфигурацией.

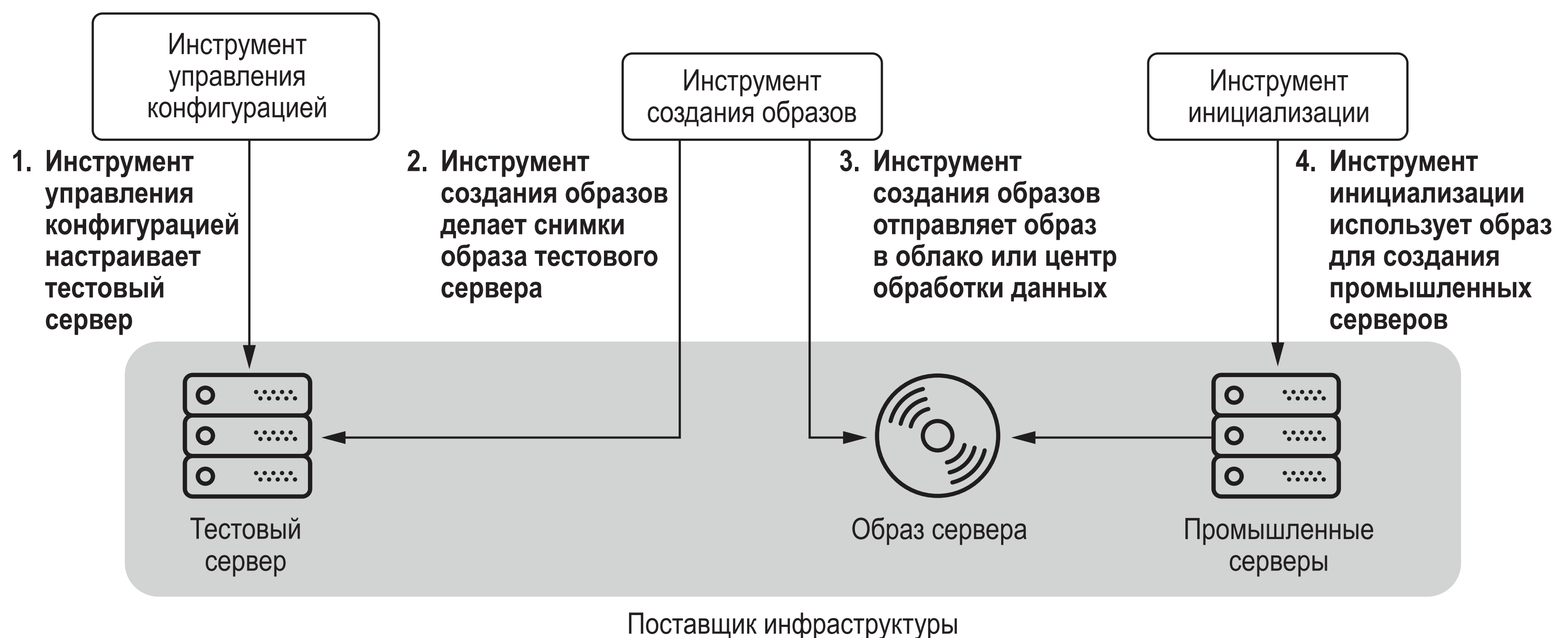


Рис. 1.14 Каждый тип инструмента IaC вносит свой вклад в жизненный цикл ресурса серверной инфраструктуры: от настройки конфигурации до получения образа и полного развертывания

Этот рабочий процесс представляет собой идеальный комплексный подход для управления и развертывания серверов с помощью инструментов IaC. Однако, как вы скоро узнаете, инфраструктура может быть сложной, и этот подход применим не к каждому варианту использования. Различия инфраструктурных систем и их зависимости в первую очередь усложняют инициализацию инфраструктуры, которую я выбрала в качестве основной темы данной книги, и именно поэтому в примерах используются инструменты инициализации.

Краткое содержание главы

- Инфраструктура может представлять собой программное обеспечение, платформу или аппаратное обеспечение, которое предоставляет или развертывает приложения в промышленной среде.
- Инфраструктура как код – это методология DevOps по автоматизации инфраструктуры для достижения надежности, масштабируемости и безопасности.
- Принципами IaC являются воспроизводимость, идемпотентность, компонентность и развиваемость.

- Следуя принципам IaC, вы можете улучшить процессы управления изменениями, сократить время, затрачиваемое на исправление вышедших из строя систем в долгосрочной перспективе, лучше обмениваться знаниями и контекстом, а также обеспечить безопасность своей инфраструктуры.
- Инструменты IaC предназначены для инициализации, управления конфигурацией и создания образов.

Разработка инфраструктуры как кода

https://t.me/it_boooks/2

Основные темы главы:

- как текущее состояние инфраструктуры влияет на ее воспроизводимость;
- обнаружение и устранение дрейфа инфраструктуры, возникающего из-за постепенных изменений;
- реализация передовых методов написания воспроизводимой инфраструктуры в виде кода.

Допустим, вы создали среду разработки для приложения hello-world. Вы постепенно развивали инфраструктуру, добавляя новые компоненты по мере необходимости. Но вот разработка подошла к концу, и вам необходимо воспроизвести конфигурацию инфраструктуры в промышленной среде, к которой пользователи смогут получить общий доступ. Для обеспечения высокой доступности также необходимо масштабировать промышленную среду на три географических региона.

Для этого нужно создать и обновить межсетевые экраны, балансировщики нагрузки, серверы и базы данных в новых сетях в промышленной среде. На рис. 2.1 показана структура среды разработки с межсетевым экраном, балансировщиком нагрузки, сервером и базой данных, а также компонентами, которые необходимо воспроизвести в промышленной среде.

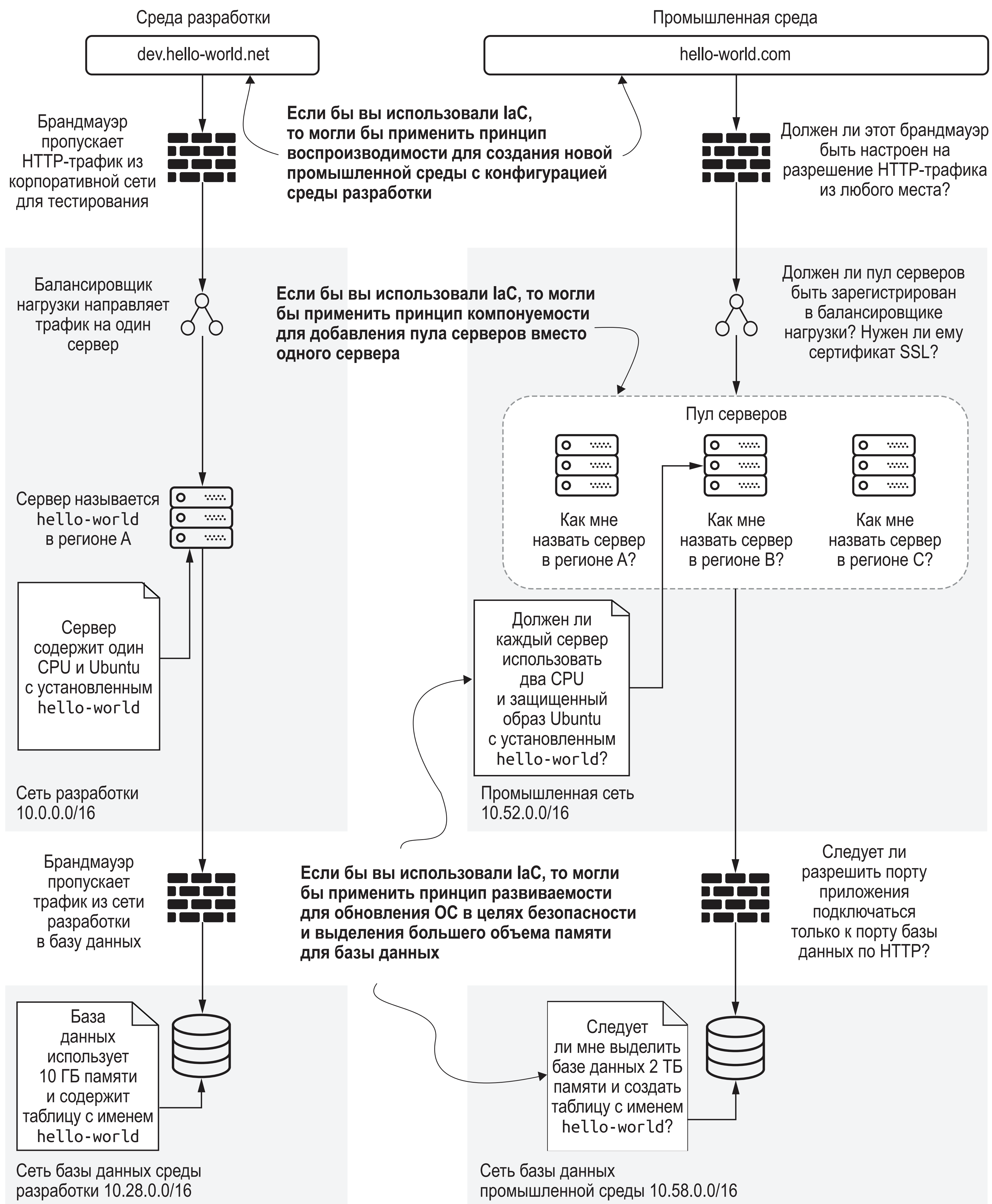


Рис. 2.1 Когда вы переносите приложение из среды разработки в промышленную среду, вам необходимо ответить на множество вопросов о конфигурациях новой инфраструктуры и провести обратный инжиниринг функциональности среды разработки

На рисунке также показаны различия между средой разработки и промышленной средой. В промышленной конфигурации должны быть три сервера для обеспечения высокой доступности, расширенные правила брандмауэра, разрешающие весь внешний HTTP-трафик, и более строгие правила брандмауэра для подключения серверов

к базе данных. После знакомства с задачей у вас может возникнуть множество вопросов о том, как лучше и проще внести изменения.

Например, вы можете задаться вопросом, почему отсутствие IaC для среды *разработки* негативно влияет на вашу способность создавать *промышленную* среду. Первая причина заключается в том, что невозможно легко воспроизвести ресурсы инфраструктуры. Вам придется потратить неделю на ручную настройку! Вместо этого с помощью IaC вы можете скопировать и вставить конфигурацию в рабочую среду, а затем внести лишь некоторые изменения для промышленной среды.

Во-вторых, вы не можете легко *компоновать* имеющиеся ресурсы инфраструктуры с новыми. Для промышленной среды вам нужен пул серверов, а не один сервер. Если вы разработали модуль для создания сервера инфраструктуры, вы можете использовать его для создания нескольких серверов, а не создавать конфигурации с нуля.

Наконец, невозможно легко *развивать* промышленную среду с учетом ее специфичных требований. Промышленная среда нуждается в более продвинутых ресурсах инфраструктуры, таких как безопасные операционные системы и более крупная база данных. Вам придется вручную настроить конфигурацию, которую вы никогда не запускали в среде разработки.

Вы можете решить эти проблемы и улучшить воспроизводимость, компоновку и развиваемость двумя способами. Во-первых, вам нужен способ переноса инфраструктуры, настроенной вручную, в IaC. Во-вторых, вам нужно написать чистую конфигурацию IaC, чтобы обеспечить воспроизводимость и развиваемость.

Первая часть этой главы посвящена фундаментальным концепциям написания IaC и переноса существующей инфраструктуры в код. Во второй части главы будет рассмотрено применение правил гигиены кода к инфраструктуре. Сочетание этих подходов поможет вам написать воспроизводимую IaC и подготовить почву для будущей компоновки и развития вашей системы.

2.1 Описание изменений инфраструктуры

В главе 1 я упоминала, что IaC автоматизирует изменения. Оказывается, воспроизведение и автоматизация множества последовательных изменений требуют усилий. Например, если вы хотите создать сервер на GCP и управлять им, то обычно выполняете следующие действия:

- 1 создаете сервер в GCP с помощью консоли, терминала или кода;
- 2 читаете метаданные сервера в GCP, чтобы убедиться, что вы создали сервер с правильными спецификациями, например Ubuntu 18.04 в качестве операционной системы;
- 3 обновляете сервер в GCP, указав открытый сетевой адрес для общего доступа;
- 4 удаляете сервер в GCP, когда он вам больше не нужен.

Чтобы выполнить более сложные обновления или воспроизвести сервер в другой среде, необходимо выполнить следующие действия:

- 1 создать сервер;
- 2 убедиться, что он существует, с помощью команды чтения метаданных;
- 3 обновить его, если нужен общий доступ;
- 4 удалить сервер, если он больше не нужен.

Независимо от того, какой ресурс вы автоматизируете, вы всегда можете разбить свои изменения на создание, чтение, обновление и удаление (create, read, update, delete – CRUD). Вы создаете ресурс инфраструктуры, ищете его метаданные, обновляете его свойства и удаляете его, когда он вам больше не нужен.

ПРИМЕЧАНИЕ В описании изменений редко говорят «прочитать метаданные сервера». Этап чтения для проверки того, что ресурс создан или обновлен, подразумевается по умолчанию.

CRUD позволяет вам автоматизировать вашу инфраструктуру шаг за шагом в определенном порядке. Этот подход, называемый *императивным стилем*, описывает, как настроить инфраструктуру. Вы можете рассматривать его как инструкцию по эксплуатации.

ОПРЕДЕЛЕНИЕ *Императивный стиль* IaC описывает пошаговую настройку ресурса инфраструктуры.

Хотя императивный стиль кажется интуитивно понятным, он не масштабируется по мере внесения новых изменений в систему. Однажды мне пришлось создавать новую среду базы данных на основе среды разработки. Я принялась вручную переносить в сценарий конфигурации 200 запросов на изменения, созданных для среды разработки за два года. Каждый запрос на изменение представлял собой серию шагов по созданию, обновлению и удалению ресурсов. На создание среды, которая все еще не соответствовала существующей среде разработки, у меня ушло полтора месяца!

Вместо того чтобы кропотливо воссоздавать каждый шаг, мне хотелось просто описать новую среду базы данных на основе текущего состояния среды разработки и переложить на инструмент все заботы по достижению этого состояния. В большинстве случаев при разработке IaC вам будет проще воспроизводить среды и вносить изменения в декларативном стиле. Декларативный стиль описывает желаемое конечное состояние ресурса инфраструктуры. Инструмент решает, какие шаги необходимо предпринять для достижения этого состояния.

ОПРЕДЕЛЕНИЕ *Декларативный стиль* IaC описывает желаемое конечное состояние ресурса инфраструктуры. Сред-

ства автоматизации и инструменты без вашего участия решают, как достичь конечного состояния.

Этот процесс с IaC осуществляется в несколько шагов. Во-первых, вам необходимо найти источники информации о серверах базы данных. Далее вы получаете IP-адреса базы данных. Наконец, вы пишете конфигурацию на основе собранной информации.

Ваша конфигурация, зафиксированная в системе контроля версий, становится *эталонной конфигурацией инфраструктуры*. Вы объявляете желаемое состояние новой среды базы данных, вместо того чтобы описывать набор шагов, которые могут и не закончиться ожидаемым результатом.

ОПРЕДЕЛЕНИЕ *Эталонная конфигурация инфраструктуры* представляет собой источник последовательной и структурированной информации о желаемом состоянии вашей системы.

Далее вы вносите последовательные изменения в эталонную конфигурацию. Однако даже в идеальных обстоятельствах (например, в случае с GitOps в главе 7) у вас, вероятно, возникнет некоторый дрейф конфигурации из-за ручных изменений. Если вы используете декларативный стиль и создаете эталонную конфигурацию, вы можете прибегнуть к принципу неизменности, чтобы снизить риск сбоя.

Упражнение 2.1

Какой стиль настройки инфраструктуры используется в следующем примере – императивный или декларативный?

```
if __name__ == "__main__":
    update_packages()
    read_ssh_keys()
    update_users()
    if enable_secure_configuration:
        update_ip_tables()
```

Ответы на упражнения приведены в приложении В.

2.2 Что такое неизменность инфраструктуры?

Как предотвратить дрейф конфигурации и быстро воспроизвести инфраструктуру? Прежде всего нужно пересмотреть ваше понимание изменений. Представьте, что вы создаете сервер с Python версии 2. Вы можете поменять свои скрипты для входа на сервер и обновить Python до версии 3 без перезапуска сервера. Это пример

изменяемой инфраструктуры, поскольку вы можете обновлять сервер «на ходу», не перезапуская его.

ОПРЕДЕЛЕНИЕ *Изменяемая инфраструктура* (mutable infrastructure) означает, что вы можете обновлять существующий ресурс инфраструктуры без его повторного создания или перезапуска.

Однако отношение к серверу как к изменяемой инфраструктуре вызывает проблему. Оказывается, другие пакеты на сервере не работают с Python 3. Вместо того чтобы вдогонку обновлять все остальные пакеты и нарушать работу сервера, вы можете разработать сценарий обновления, который создает новый сервер с Python версии 3 и совместимыми зависимостями. Затем вы можете удалить старый сервер с Python 2. На рис. 2.2 показано, как это сделать.

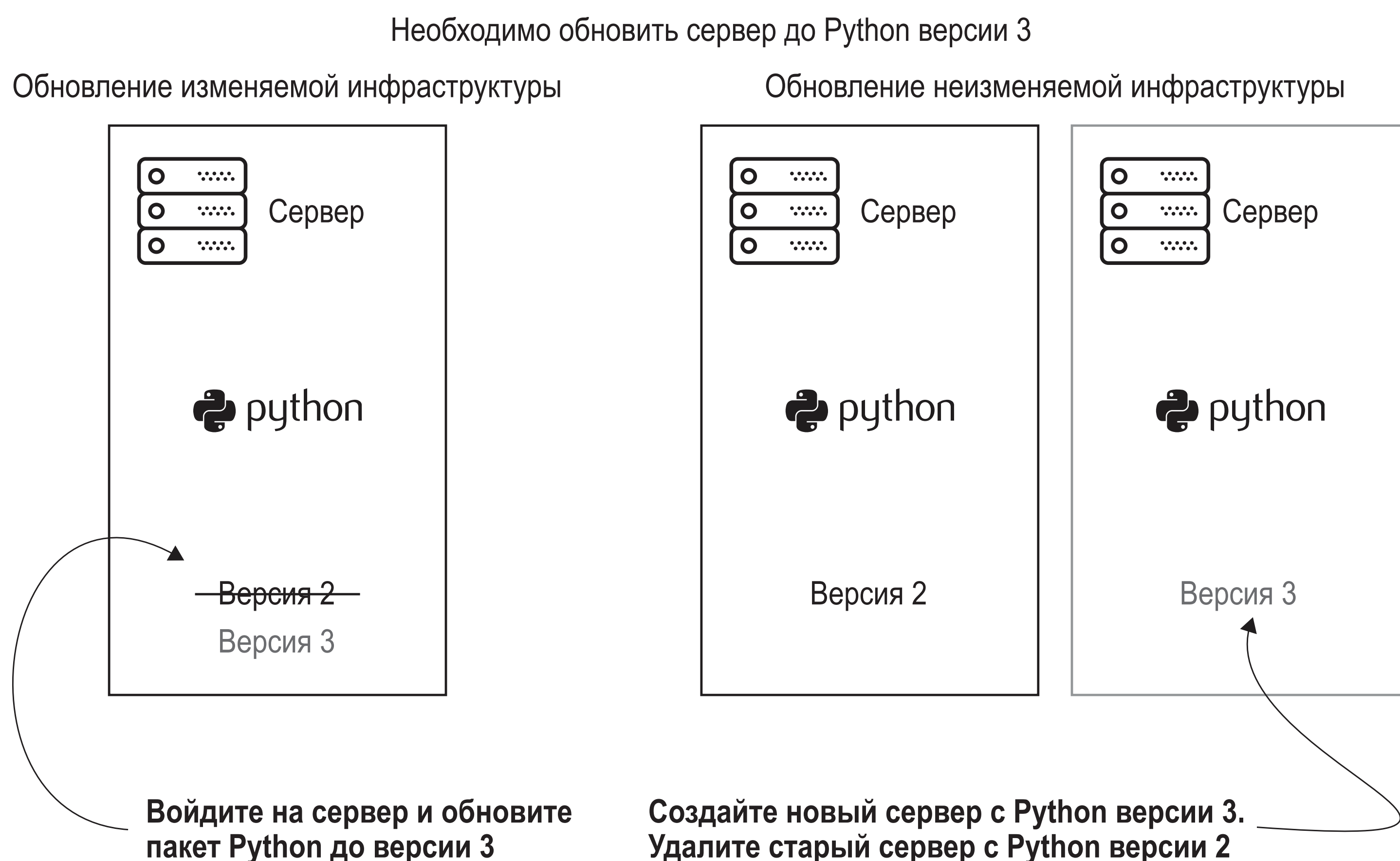


Рис. 2.2 Сервер можно изменить, войдя в систему и обновив версию пакета Python. Сравните это с принципом неизменности, когда вы полностью заменяете старый сервер новым, обновленным до Python 3

Ваши новые сценарии рассматривают сервер как *неизменяемую инфраструктуру*, в которой невозможны частичные изменения. Вы не обновляете существующую инфраструктуру. *Неизменность* (immutability) означает, что после создания ресурса вы не меняете его конфигурацию.

ОПРЕДЕЛЕНИЕ *Неизменяемая инфраструктура* означает, что вы должны создать новый ресурс для любых изменений в ее конфигурации. Вы не изменяете ресурс после его создания.

Зачем нужен второй способ обновления сервера? Некоторые обновления приведут к поломке ресурса, если сделать его изменяемым. Чтобы снизить риск сбоя, вы можете создать совершенно новый ресурс с обновлениями, запустить его и безопасно удалить старый ресурс.

Неизменность основана на последовательности операций создания и удаления. Создание нового ресурса уменьшает дрейф (разницу между фактической и ожидаемой конфигурациями), поскольку новый ресурс полностью соответствует сценарию IaC. Вы можете расширить этот подход за пределы ресурсов сервера, даже до бессерверных функций или целых инфраструктурных кластеров.

В любом случае вы создаете новый ресурс вместо обновления существующего.

ПРИМЕЧАНИЕ Построители образов машин используют концепцию неизменяемой инфраструктуры. Для любых обновлений сервера требуется новый образ машины, который создает и предоставляет построитель. Изменения сервера, такие как регистрация IP-адреса, должны передаваться в качестве параметров в сценарий запуска, определенный построителем образа.

Обеспечение неизменности влияет на способ внесения изменений. Создание нового ресурса требует соблюдения принципа воспроизводимости. Поэтому IaC хорошо подходит для обеспечения неизменности при развертывании обновлений. Например, вы можете с легкостью создавать новый брандмауэр каждый раз, когда вам нужно его обновить. Новый брандмауэр отменяет любые ручные правила, добавленные кем-то за пределами IaC, повышая безопасность системы и уменьшая дрейф.

Неизменность также повышает доступность системы и снижает вероятность сбоев в критически важных приложениях. Создание нового ресурса вместо модификации старого изолирует изменения в рамках нового ресурса, ограничивая потенциальный ущерб, если что-то пойдет не так. Подробнее об этом я расскажу в главе 9.

Однако соблюдение принципа неизменности иногда требует дополнительных трудозатрат. На рис. 2.3 сравнивается влияние изменяемой и неизменяемой инфраструктур. Рассматривая сервер как изменяемый ресурс, вы выполняете локальное обновление пакета Python. Это обновление влияет на малую часть общего состояния сервера. Но в соответствии с принципом неизменности вы заменяете сервер целиком, рискуя затронуть другие ресурсы, зависящие от этого сервера.

В таком случае обновление по принципу неизменности может занять *больше* времени, чем локальное обновление изменяемой инфраструктуры! Вы не можете рассчитывать, что вся инфраструктура

будет всегда оставаться неизменной. Если вы намерены полностью заменить десятки тысяч серверов, приготовьтесь потратить несколько дней на их повторное создание. Локальное обновление может занять всего сутки – но при условии, что вы ничего не сломаете.

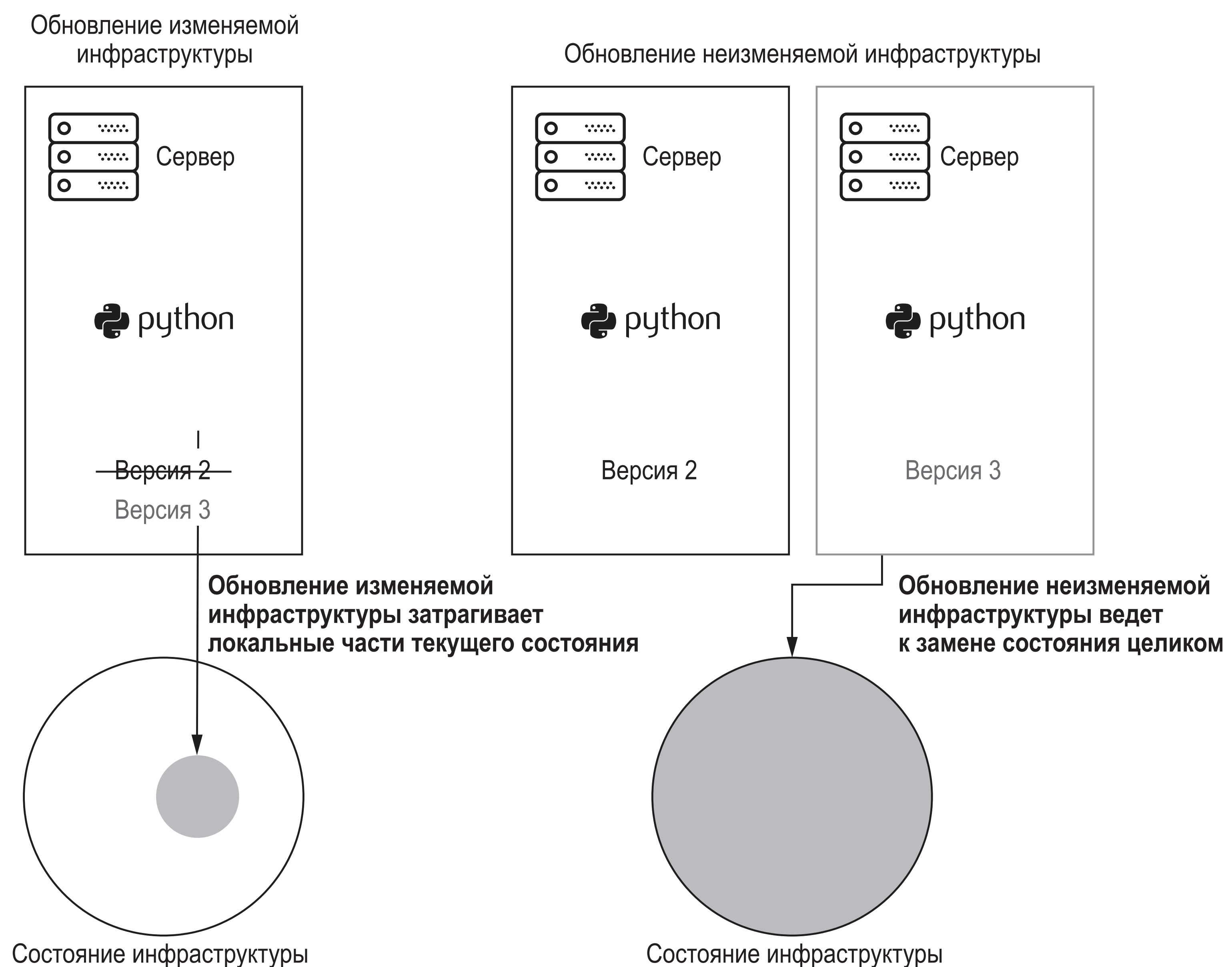


Рис. 2.3 Обновление изменяемых ресурсов влияет на небольшую часть состояния инфраструктуры, тогда как неизменяемые ресурсы необходимо создавать заново

На практике вы будете переключаться между изменяемой и неизменяемой инфраструктурами в зависимости от обстоятельств. Неизменяемая инфраструктура помогает снизить потенциальный риск сбоя в вашей системе, а изменяемая способствует более быстрым обновлениям. Инфраструктуру часто рассматривают как изменяемую, когда нужно срочно исправить систему. Как выполняется переход между изменяемой и неизменяемой инфраструктурами?

2.2.1 Внесение нештатных изменений

Иногда обновления слишком незначительны по масштабу и воздействию на систему, и было бы неразумно каждый раз создавать и развертывать новый ресурс. В таком случае вы можете прибегнуть к локальным изменениям.

Представьте, что вы и ваш друг встретились в кафе. Ваш друг заказывает капучино с соевым молоком. Однако бариста случайно

добавляет коровье молоко. Бариста вынужден приготовить новый капучино для вашего друга, потому что молоко влияет на всю чашку. Ваш друг ждет еще 5–10 минут. В отличие от друга, вы получаете чашку черного кофе и добавляете молоко и сахар по вкусу. Это занимает несколько секунд. Если сахара недостаточно, просто добавьте еще.

Вам понадобится меньше времени, чтобы обновить свой изменяемый черный кофе, по сравнению с вашим другом, пожелавшим полностью обновить неизменяемый капучино. Аналогично для обновления изменяемого ресурса требуется гораздо меньше времени, усилий и затрат. Когда вы временно рассматриваете инфраструктуру как изменяемый ресурс, вы вносите *срочные изменения* (out-of-band change).

ОПРЕДЕЛЕНИЕ *Срочное изменение* – это быстро реализуемое изменение, которое временно рассматривает неизменяемую инфраструктуру как изменяемую.

Когда вы нарушаете принцип неизменности срочным изменением, вы сокращаете время обновления, но увеличиваете риск влияния на другие изменения в будущем. После внесения срочного изменения вам необходимо исправить базовую конфигурацию в системе контроля версий, чтобы вернуться к неизменяемой инфраструктуре. В чем заключается этот процесс исправления?

После внесения срочного изменения вы должны согласовать фактическое состояние и желаемую конфигурацию. Давайте вернемся к моему примеру сервера на рис. 2.4. Сначала вы входите на сервер и обновляете Python до версии 3. Затем вы изменяете конфигурацию в системе управления версиями, поэтому на все новые серверы будет установлен Python версии 3. Теперь конфигурация сервера совпадает с базовой конфигурацией в системе управления версиями.

Почему необходимо обновлять конфигурацию IaC в системе контроля версий сразу после внесения срочных изменений? В главе 1 было сказано, что ручные изменения отрицательно влияют на воспроизводимость. При переносе срочных изменений в будущую неизменяемую инфраструктуру сохраняется воспроизводимость. После регистрации срочного изменения в базовой конфигурации IaC вы сможете повторно развертывать это изменение на любом сервере с гарантией точной воспроизводимости. Такое поведение соответствует принципу идемпотентности!

Если вы будете часто вносить небольшие срочные изменения, вам придется без конца исправлять базовую конфигурацию. В таком случае следует отдать приоритет неизменности, чтобы обеспечить воспроизводимость. В моем примере с кофе бариста всегда может заменить напиток, даже если вы случайно опрокинете сахарницу в свой изменяемый кофе. Я рекомендую разработать процедуры внесения изменений в вашей организации, чтобы ограничить частоту сроч-

ных изменений и обеспечить соответствие обновлений конфигурации в IaC. Вы всегда можете вернуться к использованию неизменяемой конфигурации инфраструктуры, чтобы исправить неудачное срочное изменение.

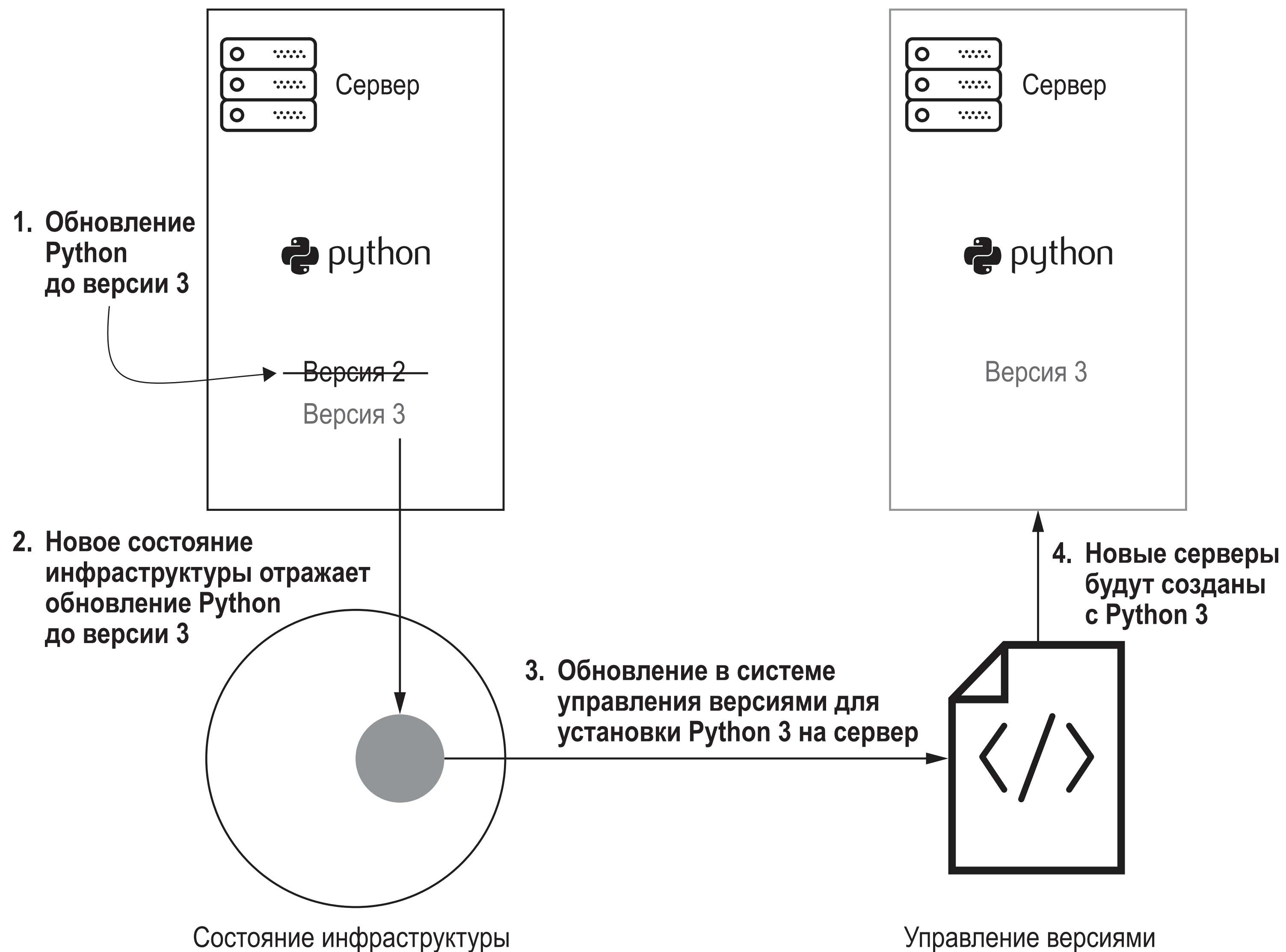


Рис. 2.4 После обновления изменяемого ресурса необходимо произвести изменения в системе контроля версий, чтобы учесть срочные изменения

Упражнение 2.2

Какие из следующих изменений выигрывают от принципа неизменности (выберите все подходящие варианты):

- a** сокращение сети для уменьшения количества IP-адресов;
- b** добавление столбца в реляционную базу данных;
- c** добавление нового IP-адреса в существующую запись DNS;
- d** обновление пакетов сервера до обратно несовместимых версий;
- e** миграция ресурсов инфраструктуры в другой регион.

Ответы на упражнения даны в приложении В.

2.2.2 Переход к технологии IaC

Принцип неизменности IaC позволяет с помощью системы контроля версий поддерживать в актуальном состоянии источник достоверной информации об инфраструктуре и облегчать будущее воспроизведение. Фактически соблюдение принципа неизменности

означает, что вы постоянно создаете новые ресурсы. Это хорошо работает для *новых сред* (greenfield), в которых нет активных ресурсов.

Однако в большинстве организаций имеются *давно работающие среды* (brownfield) с активными серверами, балансировщиками нагрузки и сетями. Напомню, что в примере этой главы мы рассматриваем существующую среду разработки под названием hello-world. Вы вошли в панель управления своего провайдера инфраструктуры и вручную создали набор ресурсов.

В целом в существующей среде инфраструктура рассматривается как изменчивая. Вам нужен способ перехода от практики ручного обновления изменяемой инфраструктуры к автоматическому обновлению неизменяемой IaC. Как перевести ресурсы инфраструктуры среды в неизменяемый режим?

Давайте перенесем среду разработки hello-world в неизменяемый IaC. Прежде чем начать, составьте список ресурсов инфраструктуры в среде. Он содержит сети, серверы, балансировщики нагрузки, брандмауэры и записи системы доменных имен (DNS).

Базовая инфраструктура

Для начала найдите базовый ресурс инфраструктуры, который должны использовать другие ресурсы. Например, каждый ресурс инфраструктуры зависит от сети в среде разработки. Начните писать сценарий IaC для базы данных и сетей в среде разработки, поскольку их используют сервер, балансировщик нагрузки и база данных. Вы не можете приступить к переносу ресурсов, работающих в сетях, пока сети не описаны в виде кода.

На рис. 2.5 показано, как вы используете терминал для доступа к интерфейсу прикладного программирования (API) провайдера инфраструктуры. Команда терминала выводит имя и диапазон IP-адресов (блок бесклассовой междоменной маршрутизации, classless inter-domain routing, CIDR) базы данных и сетей разработки. Далее вы заново конструируете сеть, копируя имя и блок CIDR каждой сети в IaC.

Зачем выполнять разбор конфигурации и воспроизведение сетей в IaC? Вы должны точно сопоставить IaC с фактическим состоянием сетевого ресурса. Если у вас возникнет расхождение конфигурации с реальностью, называемое *дрейфом*, ваш сценарий IaC может случайно сломать сеть (и все, что к ней подключено)!

Если возможно, импортируйте ресурс в состояние IaC. Ресурс уже существует, и вам нужно, чтобы ваш инструмент инициализации это обнаружил. На этапе импорта существующий ресурс переносится в систему управления IaC. Чтобы завершить перенос сетевых ресурсов, снова запустите IaC и убедитесь, что дрейф отсутствует.

Многие инструменты инициализации инфраструктуры имеют функцию импорта ресурсов. Например, CloudFormation использует команду `resource import`. Аналогично Terraform предлагает команду `terraform import`.

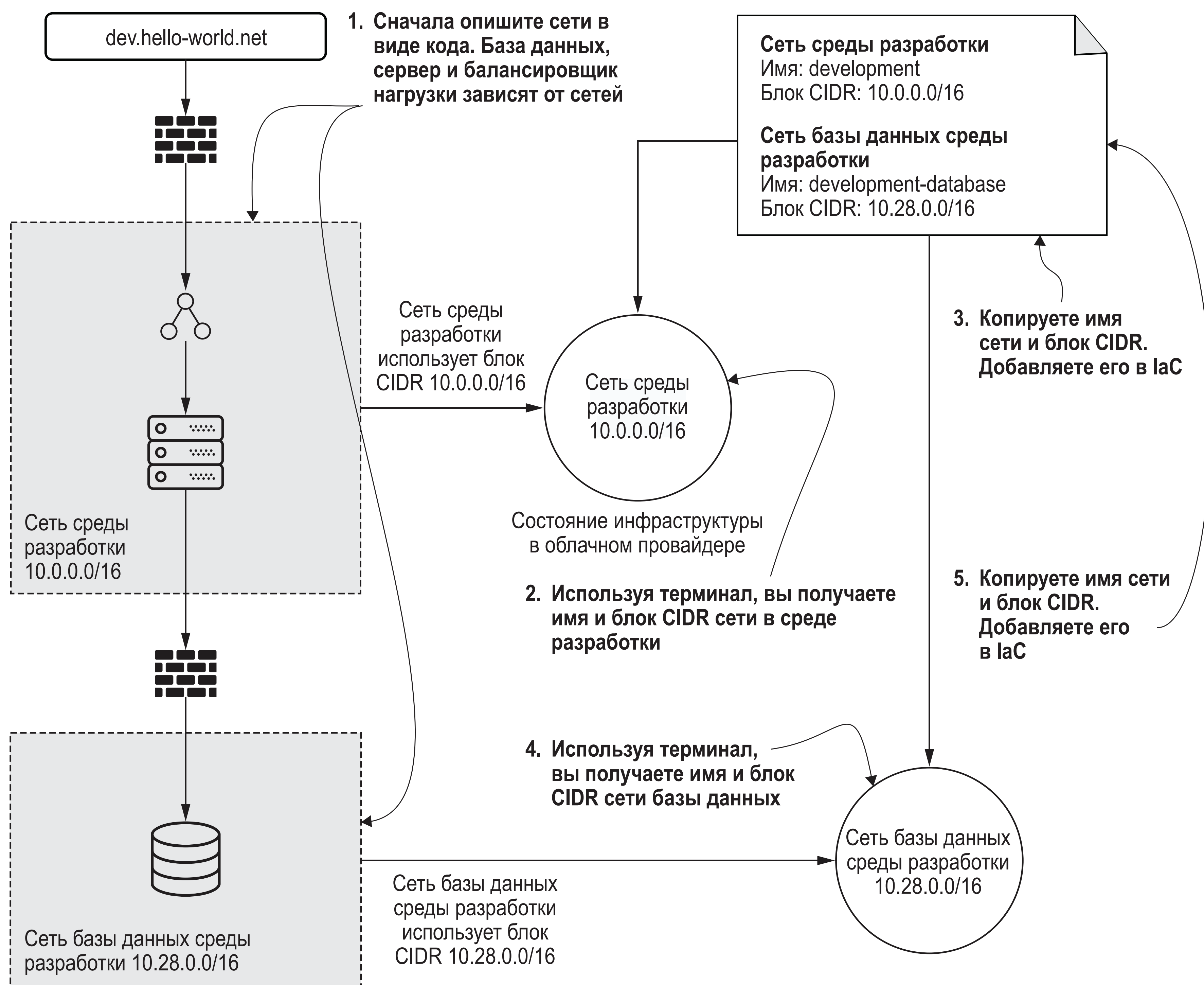


Рис. 2.5 Сначала выполните обратный инжиниринг сетей для базы данных и сервера и запишите их конфигурации в виде кода

Если вы разрабатываете IaC без инструмента инициализации инфраструктуры, вам не нужна возможность прямого импорта. Вместо этого вы пишете код для создания нового ресурса. Иногда проще использовать воспроизводимость для создания совершенно нового ресурса. Если вам сложно создать новый ресурс, напишите код с условными операторами, которые проверяют, существуют ли ресурсы.

На рис. 2.6 показан весь рабочий процесс принятия решения о повторной разработке сети и о том, можете ли вы использовать инструмент инициализации инфраструктуры для перевода вашего ресурса в неизменяемый режим.

На этой диаграмме представлены соображения по созданию новых ресурсов или написанию условных операторов для существующих ресурсов. Во время миграции вам необходимо несколько раз запустить IaC, чтобы убедиться в отсутствии дрейфа.

Почему процесс принятия решений о переходе к неизменяемой структуре так сложен? Все дело в необходимости придерживаться принципов воспроизводимости, идемпотентности и компонентности. Вы должны максимально точно воспроизвести ресурсы в IaC.

Если вы не можете импортировать ресурс, вы можете хотя бы воспроизвести новый.

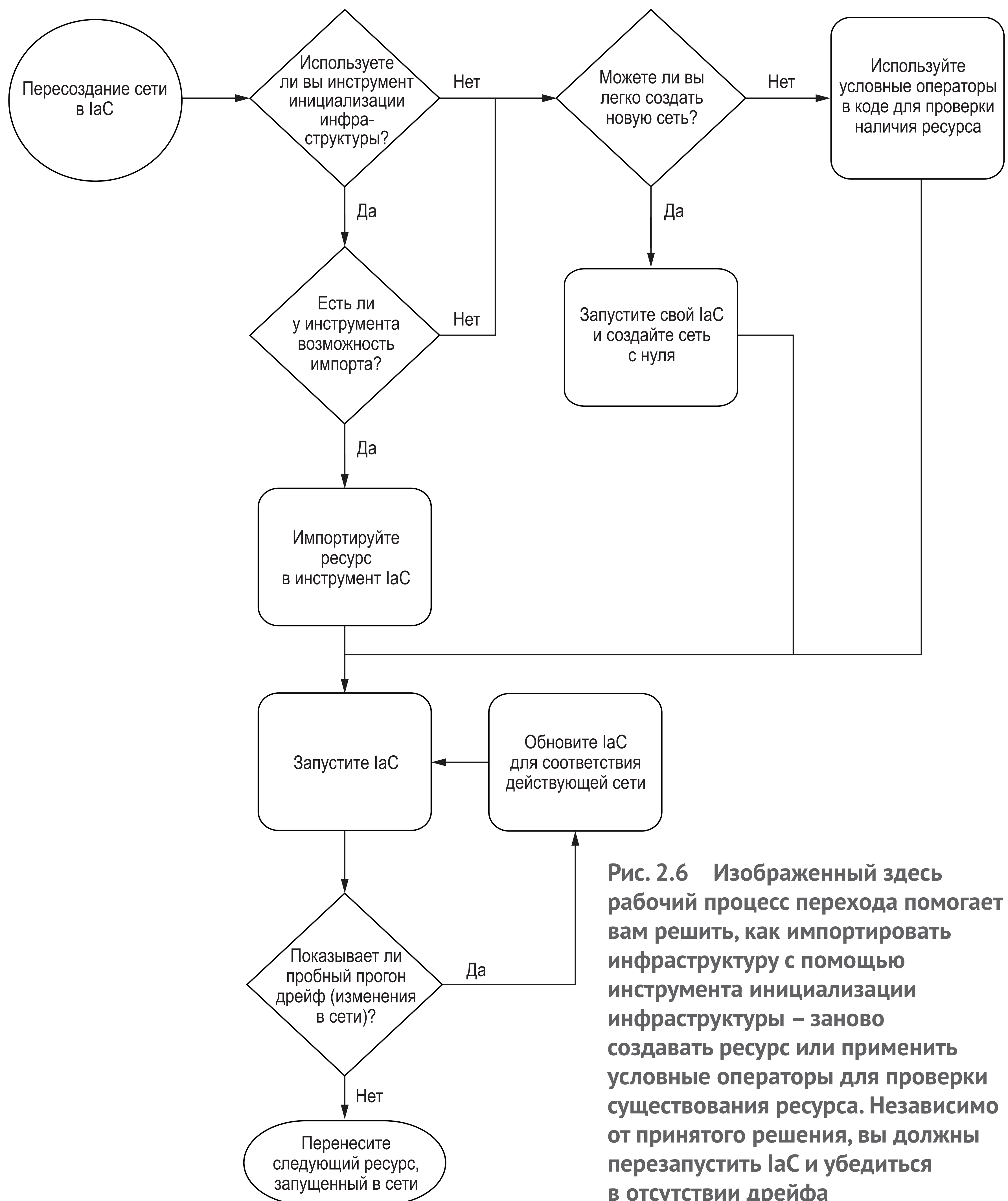


Рис. 2.6 Изображенный здесь рабочий процесс перехода помогает вам решить, как импортировать инфраструктуру с помощью инструмента инициализации инфраструктуры – заново создавать ресурс или применить условные операторы для проверки существования ресурса. Независимо от принятого решения, вы должны перезапустить IaC и убедиться в отсутствии дрейфа

Более того, при повторном запуске кода применяется принцип идемпотентности, который гарантирует, что вы не создадите ресурс заново (если в этом нет необходимости). Если вы устранили дрейф, идемпотентность не должна изменить активную сеть. Аналогично

принцип компонуемости позволяет вам переносить каждый ресурс по отдельности, чтобы избежать нарушения работы системы.

Работая над другими ресурсами, помните о процессе принятия решений. Вы можете применить его к каждому ресурсу, который вы переносите в IaC, пока не завершите миграцию. Мы вернемся к частям этого процесса принятия решений при рефакторинге IaC в главе 10.

Ресурсы, зависящие от базовой инфраструктуры

После переноса базовой сетевой инфраструктуры можно приступать к работе с серверами и другими компонентами. И снова вы используете терминал для получения атрибутов сервера `hello-world`. Он работает в регионе А с операционной системой Ubuntu и одним процессором. Вы записываете спецификацию сервера в его конфигурации, отмечая его зависимость от сети в среде разработки. Точно так же через терминал вы узнаете, что база данных использует 10 ГБ памяти. Вы копируете этот параметр в IaC и отмечаете, что он использует сеть базы данных в среде разработки. На рис. 2.7 показан процесс переноса конфигурации сервера и базы данных в сценарий IaC.

Теперь вам нужно перенести второй набор ресурсов, использующих сеть. Используйте принцип компонуемости, чтобы изолировать эти ресурсы инфраструктуры и выполнять итеративные обновления. Небольшие изменения на следующих уровнях инфраструктуры помогают предотвратить более крупные сбои системы. В главе 7 подробнее говорится о развертывании небольших изменений в инфраструктуре.

Прежде чем перейти к следующему набору ресурсов, завершите цикл переноса, запустив IaC и проверив наличие дрейфа. Убедитесь, что сеть, сервер и база данных не требуют изменений в своем IaC. После устранения всех расхождений вы можете перейти к оставшимся ресурсам (DNS, правила брандмауэра и балансировщики нагрузки).

Наконец, на рис. 2.8 показан перенос оставшейся конфигурации DNS, правил брандмауэра и балансировщиков нагрузки. Они зависят от существующей конфигурации серверов и баз данных. Никакие другие ресурсы от них не зависят.

Зачем проходить через столь кропотливый и запутанный процесс переноса различных уровней инфраструктуры? В вашей существующей среде не было стабильной эталонной конфигурации инфраструктуры, поэтому вам необходимо ее создать. Завершив добавление ресурсов инфраструктуры в конфигурацию, вы создаете и фиксируете эталонное состояние, от которого отталкиваетесь в дальнейшем. Наличие эталонной конфигурации IaC позволяет относиться к существующей среде как к неизменной инфраструктуре.

В целом вы всегда будете продвигаться к неизменяемости, начиная с базовых ресурсов, а затем переноса ресурсы верхнего уровня. Начиная перенос, первым делом определите базовые ресурсы ниж-

него уровня, от которых зависит все остальное. Опишите эти низкоуровневые ресурсы, такие как сети, учетные записи или проекты, а также IAM, в коде IaC.

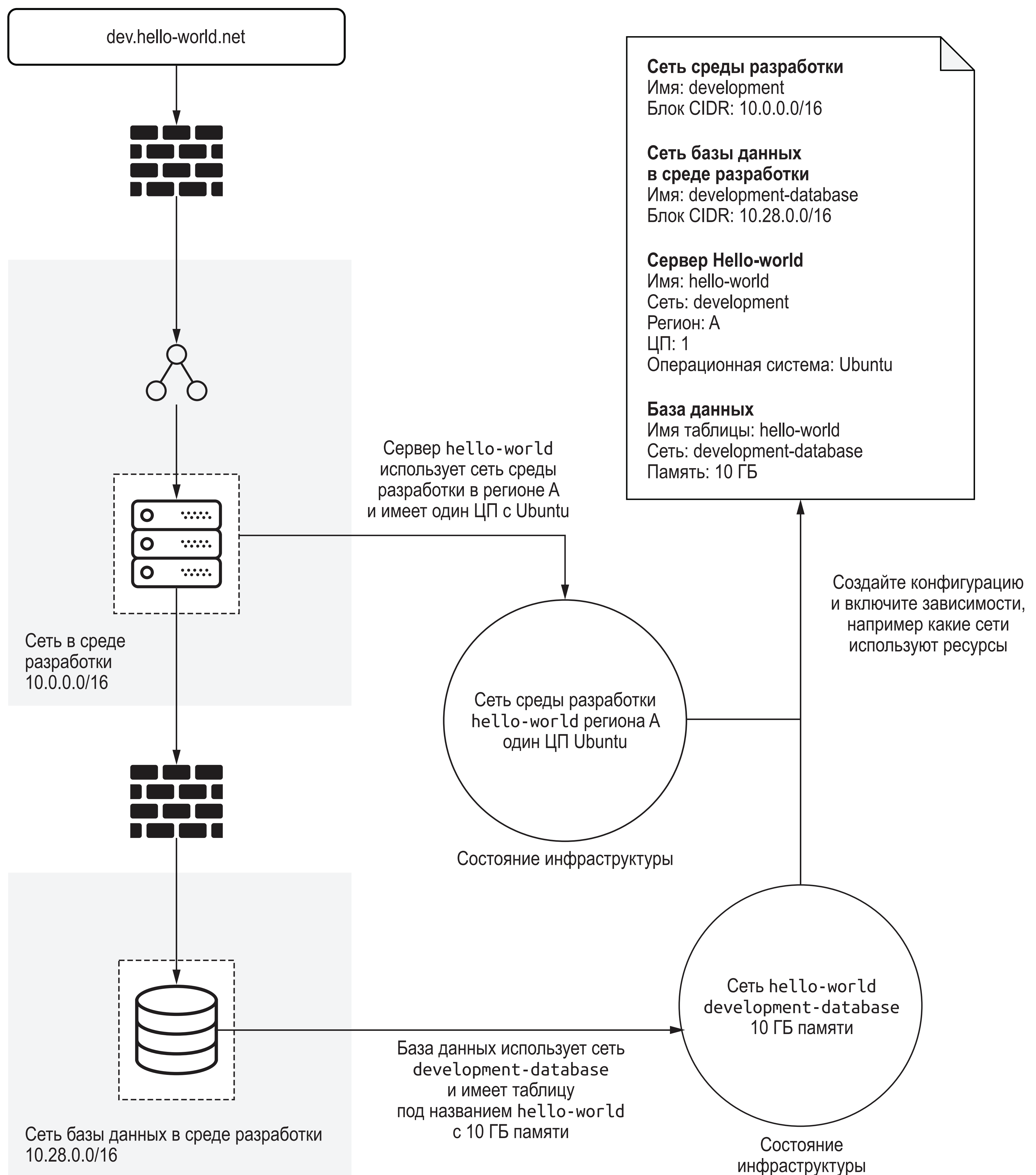


Рис. 2.7 После миграции базовой инфраструктуры, такой как сети, перенесите ресурсы сервера и базы данных. Они зависят от базовой инфраструктуры, но не зависят друг от друга

Затем принимайтесь за ресурсы более высокого уровня, такие как серверы, очереди или базы данных. Брандмауэры, балансировщики нагрузки, DNS и оповещения зависят от существования серверов,

очереди и баз данных. В конце процесса вы можете перенести ресурсы с наименьшим количеством зависимостей. Мы подробнее обсудим инфраструктурные зависимости в главе 4.

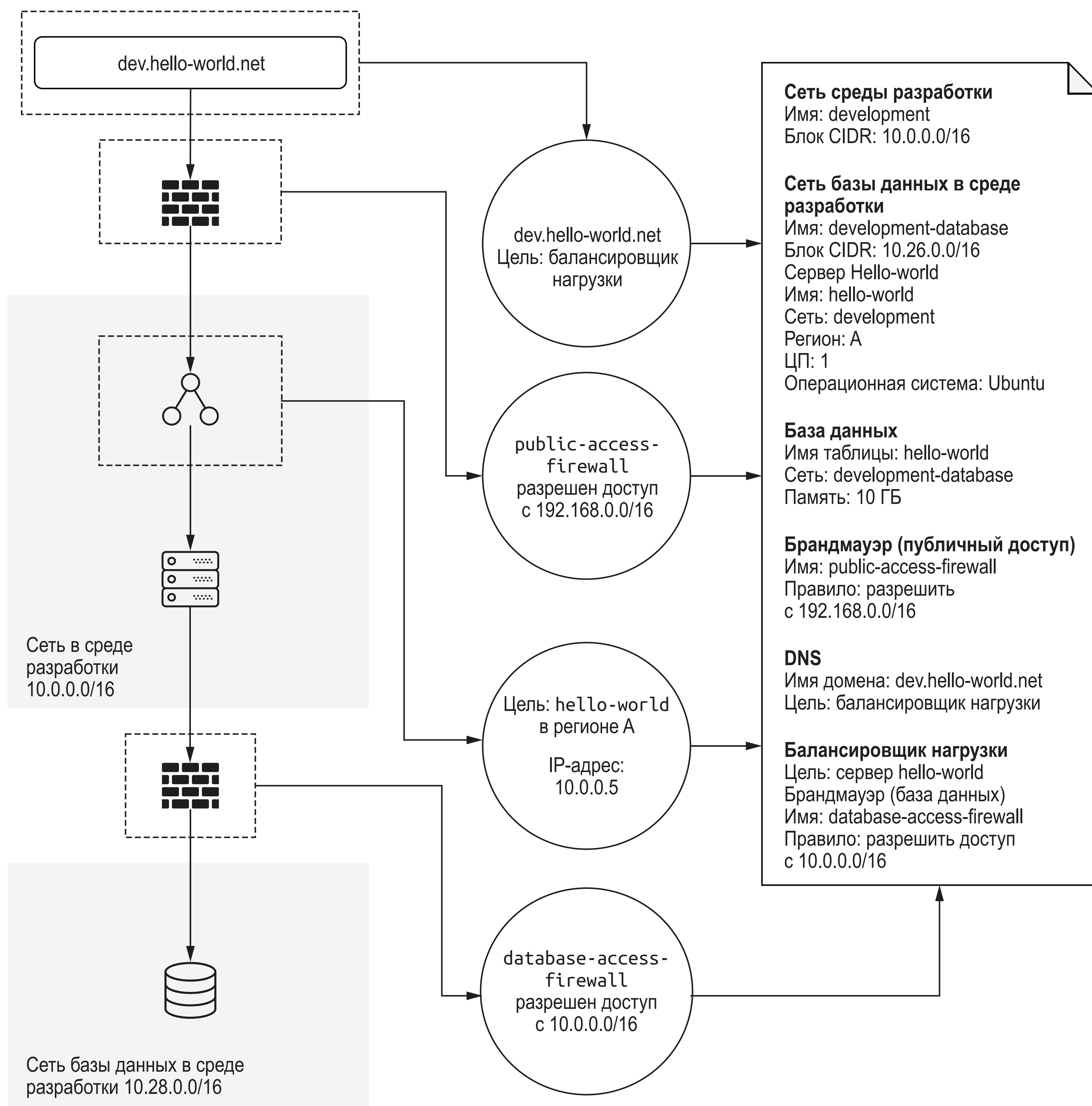


Рис. 2.8 Наконец, перенесите ресурсы с наименьшим количеством зависимостей или требующие настройки сервера и базы данных

ПРИМЕЧАНИЕ Граф зависимостей представляет зависимости между ресурсами инфраструктуры. Инструменты IaC, такие как Terraform, используют это представление для структурированного применения изменений. При переносе ресурсов вы восстанавливаете граф зависимостей. Вы можете изучить инструменты, которые будут отображать для вас состояние активной инфраструктуры и выделять зависимости, чтобы упростить задачу.

Этапы миграции

Обычно я выполняю следующие шаги для оценки зависимостей и структурированного переноса существующих ресурсов в IaC:

- 1 Перенесите *начальный вход в систему, учетные записи и конструкции изоляции ресурсов провайдера*. Например, я начинаю с конфигурации для учетной записи облачного провайдера или проекта и своей первоначальной учетной записи службы для автоматизации.
- 2 Перенесите *сети, подсети, маршрутизацию и конфигурацию корневого DNS*, если они есть. Конфигурация корневого DNS может включать сертификаты Secure Sockets Layer (SSL). Например, я создала корневой домен `hello-world.net` и его сертификат SSL для подготовки к созданию поддоменов, таких как `dev.hello-world.net`.
- 3 Перенесите *вычислительные ресурсы*, такие как серверы приложений или базы данных.
- 4 Перенесите *платформу оркестрации вычислений и ее компоненты*, если вы используете платформу оркестрации вычислений. Например, я переношу свой кластер Kubernetes, чтобы распределять рабочие нагрузки между серверами.
- 5 Если вы используете платформу оркестрации вычислений, перенесите *развертывания приложений на платформе оркестрации вычислений*. Например, я переношу конфигурацию приложения `hello-world`, развернутого в Kubernetes.
- 6 Перенесите *очереди сообщений, кеша или платформы потоковой передачи событий*. Эти службы имеют зависимости от приложений, которые нужно разрешить, прежде чем приступать к переносу. Например, я пишу конфигурацию очереди сообщений для связи между `hello-world` и другим приложением.
- 7 Перенесите *поддомены DNS, балансировщики нагрузки и брандмауэры*. Например, я заново создаю конфигурацию правил брандмауэра между моим приложением `hello-world` и его базой данных.
- 8 Перенесите *службы оповещений или мониторинга*, связанные с ресурсами. Например, я переношу конфигурацию, которая уведомляет меня в случае сбоя приложения `hello-world`.
- 9 Наконец, выполните перенос *ресурсов SaaS*, таких как средства обработки данных или репозитории, которые не зависят от приложений. Например, это может быть задание по преобразованию данных в GCP, которое имеет особую зависимость от базы данных.

После каждого шага проверяйте правильность переноса исходных ресурсов, перезапуская конфигурацию. Редко удастся получить все необходимые параметры и зависимости с первой попытки.

ПРИМЕЧАНИЕ Согласно принципу идемпотентности, повторный запуск перенесенной конфигурации не должен изменять существующую инфраструктуру. Вам следует повторно применить конфигурацию и проверить результат. Наличие изменений означает, что ваша конфигурация неточно отражает фактическое состояние ресурса.

Если вы запустили конфигурацию и она сообщает о наличии изменений, вам необходимо исправить конфигурацию! Зачастую приходится действовать методом проб и ошибок. Поэтому я рекомендую вам протестировать каждый набор ресурсов.

Переход к неизменяемости превращается в упражнение по уменьшению дрейфа. Этот процесс демонстрирует экстремальную ситуацию, когда конфигурация отклоняется далеко от текущего состояния. Вы работаете над согласованием эталонной конфигурации инфраструктуры с реальностью, обновляя ее в системе контроля версий. Процесс импорта существующих ресурсов в новую эталонную конфигурацию применим и к рефакторингу IaC, который мы обсудим в главе 10.

2.3 Написание чистого кода

Помимо неизменяемости, еще одним способом повышения воспроизводимости является написание чистого кода конфигурации. *Гигиена кода* – это набор приемов, направленных на улучшение читаемости и структуры кода.

ОПРЕДЕЛЕНИЕ *Гигиена кода* – это набор приемов и стилей, повышающих читаемость и удобство сопровождения кода.

Гигиена IaC помогает сэкономить время, когда вам нужно повторно использовать конфигурацию. Я часто обнаруживаю, что конфигурацию инфраструктуры копируют, вставляют и редактируют с жестко закодированными значениями. Жестко закодированные значения ухудшают читаемость и воспроизводимость кода. Поскольку многие приемы написания чистого кода связаны с разработкой программного обеспечения, я предлагаю более подробно рассмотреть некоторые аспекты, специфичные для инфраструктуры.

2.3.1 Передача контекста в систему контроля версий

Как эффективно использовать систему контроля версий для обеспечения воспроизводимости? Структурированные методы управления версиями помогут вам быстро воспроизвести конфигурацию и внести обоснованные изменения. Например, вы можете обновить

правило брандмауэра, которое разрешает трафик из `app-network` в `shared-services-network`. Вы сопровождаете коммит сообщением, в котором говорится, почему вы добавили разрешение:

```
$ git commit -m "Разрешает приложению подключаться  
к очередям app-network в shared-services-network,  
поскольку приложение использует очереди.  
Все порты должны быть открыты."
```

Через несколько недель вы воспроизводите сеть в рабочей среде. Однако вы забыли, почему добавили разрешение. Просматривая историю коммитов, вы находите свое сообщение. Теперь вы знаете, что нужно приложению для доступа к очередям.

Когда вы пишете сообщения коммита для IaC, вам не нужно объяснять конфигурацию. Изменение уже отражает то, какой будет конфигурация. Вместо этого используйте сообщение коммита, чтобы объяснить, *почему* вы хотите внести изменения и *как* они повлияют на другую инфраструктуру.

ПРИМЕЧАНИЕ В этой книге я рассматриваю практики работы с системой контроля версий, предназначенные для IaC. Чтобы узнать больше о контроле версий, ознакомьтесь с руководством *Git Getting Started – About Version Control* по адресу <http://mng.bz/pOBR>. Дополнительную информацию о написании хороших сообщений о коммитах можно найти в разделе *Distributed Git – Contributing to a Project* по адресу <http://mng.bz/OoMj>. Контент для обоих разделов взят из книги *Pro Git* (Scott Chacon, Ben Straub; Apress, 2014).

Вы можете столкнуться с требованием аудита добавлять префикс номера проблемы или заявки перед сообщением коммита для отслеживания выполнения. Допустим, вы отработываете заявку `TICKET-002`. Она содержит запрос на разрешение трафика между приложением и общими службами. Чтобы связать заявку с вашим коммитом, вы добавляете номер заявки в начало сообщения:

```
$ git commit -m "TICKET-002 Разрешает приложению подключаться  
к очередям app-network в shared-services-network,  
поскольку приложение использует очереди.  
Все порты должны быть открыты."
```

Добавление информации о номере проблемы или заявки в сообщения коммита упрощает отслеживание изменений. Конфигурация становится документацией изменений, поскольку она является источником достоверной информации о ресурсах инфраструктуры. Система контроля версий также становится механизмом документирования изменений. Вы можете восстановить историю изменений и воспроизвести среды, изучив репозиторий контроля версий и конфигурацию.

2.3.2 Линтинг и форматирование

Прежде чем создать коммит, необходимо выполнить *линтинг* кода – проверку его оформления и форматирования. Сценарии IaC часто не выполняются из-за того, что вы использовали неправильное имя поля или пропустили пробелы. Неправильное имя поля может вызвать ошибку выполнения. Неправильно выровненный код часто приводит к неправильному чтению или пропуску строки конфигурации.

Представьте, что вы настраиваете сервер и ему нужно поле `ip_address`. Вместо этого вы называете поле `ip`, а позже обнаруживаете, что не можете создать сервер с помощью IaC. Как убедиться, что вы указали правильное имя поля `ip_address`?

Линтинг применяется для анализа вашего кода и выявления нестандартных или неправильных конфигураций. Большинство инструментов обладают возможностью проверки конфигурации или кода. Линтинг по имени `ip_address` обнаруживает неправильное имя поля `ip` на ранних этапах разработки.

ОПРЕДЕЛЕНИЕ *Линтинг* автоматически проверяет стиль вашего кода на наличие нестандартной конфигурации.

Зачем искать нестандартную или неправильную конфигурацию? Вам нужно убедиться, что вы написали правильную конфигурацию и не допустили критическую ошибку синтаксиса. Если в инструменте нет функции линтинга, вы всегда можете найти расширение, разработанное сообществом, или написать свои собственные правила линтинга на удобном языке программирования. Вы должны включить в линтинг проверки на соответствие стандартам безопасности, например отсутствие секретов, отправляемых в систему контроля версий (глава 8).

Помимо линтинга, вы можете использовать *форматирование* для проверки пробелов и форматов конфигурации. Соблюдение стандартов форматирования кода может показаться очевидным для разработчиков программного обеспечения, но в IaC оно становится более важным.

ОПРЕДЕЛЕНИЕ Инструмент форматирования автоматически выравнивает ваш код с учетом правильных интервалов и формата конфигурации.

Большинство инструментов используют предметно-ориентированные языки (*domain-specific language*, DSL), которые представляют собой надстройку с более высоким уровнем абстракции над обычным языком программирования. DSL проще для освоения и применения, если вы не знаете язык программирования. Эти языки используют форматы данных YAML или JSON с особыми требова-

ниями к формату. Вот почему очень полезно иметь инструменты для проверки форматирования, например не пропустили ли вы пробел в файле YAML!

Вы также можете добавить хуки (перехватчики) системы контроля версий для запуска проверок форматирования перед отправкой коммита. Допустим, вы создали ресурсы инфраструктуры с помощью CloudFormation в формате данных YAML. Для проверки полей и значений ресурсов инфраструктуры вы используете AWS CloudFormation Linter (<http://mng.bz/YGrj>). Вы также форматируете файл YAML с помощью средства форматирования шаблонов AWS CloudFormation (<http://mng.bz/GEVA>).

Вместо того чтобы каждый раз вводить команды линтинга и форматирования, вы можете добавить их в хук Git перед коммитом. Каждый раз, когда вы выполняете команду `git commit`, она проверяет правильность конфигурации и формата, прежде чем отправить код в репозиторий. Вы также можете добавить линтинг и форматирование в рабочий процесс непрерывной поставки, о котором я расскажу в главе 7.

2.3.3 Именованние ресурсов

Когда код IaC по совместительству становится документацией, вашим ресурсам, конфигурации и переменным нужны описательные имена. Однажды я создала правило брандмауэра, чтобы что-то проверить, и назвала его `firewall-rule-1`. Через две недели, когда я захотела воспроизвести конфигурацию в промышленной среде, я не смогла вспомнить, зачем создала правило в среде разработки.

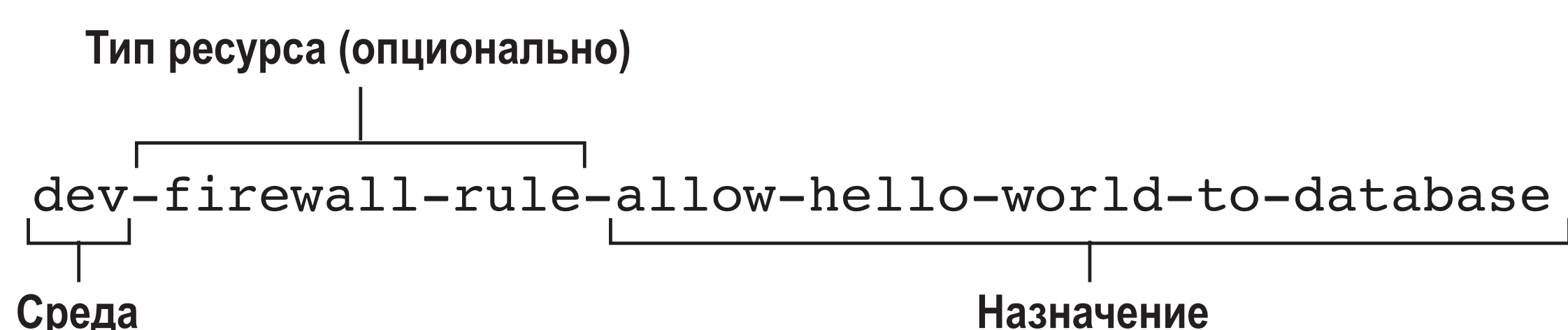


Рис. 2.9 Имя ресурса должно включать среду, тип и назначение

Почему имена должны быть такими сложными? Вам и вашим коллегам нужна возможность быстро идентифицировать ресурс для устранения неполадок, совместного использования и аудита. Наличие префикса среды гарантирует, что вы настроите ее правильно (и не вмешаетесь случайно в промышленную среду). С помощью суффикса вы сообщаете другим и напоминаете себе, что делает ресурс.

При желании вы можете указать тип ресурса. Но я обычно обхожусь без типа ресурса в имени, поскольку определяю его по метаданным ресурса. Отказ от типа ресурса помогает уложиться в количество символов, установленное вашим облачным провайдером. Дополнительную информацию о назначении или типе вашего ресурса всегда можно включить в теги ресурса (глава 8).

Опишите ресурс кому-нибудь другому

Когда я даю название ресурсу, то пытаюсь описать его кому-то другому. Если другой человек поймет назначение ресурса по короткому имени, это хорошо. Однако я помню, что если кому-то понадобится задать дополнительные вопросы о среде или типе ресурса, ответы должны быть у него под рукой.

При таком подходе получаются очень длинные имена, но я стараюсь сделать их более описательными. Распознавание назначения ресурса по его имени экономит драгоценное время на починку среды.

Помимо имен ресурсов, вам также необходимо сделать как можно более информативными имена переменных и названия конфигураций. Большинство провайдеров инфраструктуры имеют определенные имена атрибутов ресурсов. AWS называет IP-адрес сети `Cidr-Block`, а Azure называет его `address_space`.

Я стараюсь использовать систему именования конкретного провайдера, чтобы облегчить поиск документации провайдера для последующих изменений и воспроизведения. Если я переименую конфигурацию Azure в `cidr_block`, мне нужно не забыть впоследствии преобразовать параметр в `address_space`, чтобы Azure мог его использовать. В целом нужно помнить о необходимости преобразования имен полей, переменных или конфигураций при переходе на другого провайдера либо окружение.

2.3.4 Переменные и константы

Помимо именованных переменных, как узнать, какие значения должны быть изменяемыми? Допустим, приложение `hello-world` всегда обслуживает порт 8080. Вы не планируете часто менять порт, поэтому в начале настройки устанавливаете для него значение `application_port = 8080`. Однако вы жестко кодируете имя `hello-world` непосредственно в атрибутах `name` ресурсов вашей инфраструктуры.

Год спустя вы воспроизводите среду для новой версии `hello-world` на порту 3000. Вам нужно новое значение имени `hello-world-v2`. Вы обновляете `application_port` в начале вашей конфигурации до значения 3000. Хранение номера порта в переменной позволяет вам ссылаться на `application_port` во всей вашей конфигурации и сохранять значение в одном месте. Вы поздравляете себя с тем, что вам не нужно искать и заменять все значения 8080 в вашей конфигурации. Однако вы тратите час на поиск всех упоминаний `hello-world` в конфигурации вашей инфраструктуры, чтобы изменить это имя.

В этом примере у вас есть два типа входных данных. Переменная хранит значение, на которое ссылается конфигурация инфраструк-

туры. Большинство значений инфраструктуры лучше всего хранить в переменных, и на них можно ссылаться в конфигурации.

ОПРЕДЕЛЕНИЕ *Переменная* хранит значение, на которое ссылается конфигурация инфраструктуры. Будьте готовы изменять значение переменной каждый раз, когда создаете новый ресурс или среду.

В нашем примере вам следовало рассматривать в качестве переменной имя приложения `hello-world`, поскольку оно будет меняться в зависимости от среды, версии или цели. Номер порта, как правило, не меняется в зависимости от среды или цели. Константная переменная задает общее значение для набора ресурсов и редко меняется в зависимости от среды или цели.

ОПРЕДЕЛЕНИЕ *Константная переменная* (`constant variable`) устанавливает общее значение для всей конфигурации инфраструктуры. Такие переменные меняются редко.

Принимая решение о том, когда сделать значение конфигурации обычной переменной, а когда константой, учитывайте влияние и последствия изменения ее значения для безопасности. Частота изменений не столь важна. Если изменение значения влияет на зависимости инфраструктуры или ставит под угрозу конфиденциальную информацию, установите его как переменную. Вы всегда должны задавать имена или среды в качестве переменных.

В отличие от разработки программного обеспечения, которая требует меньшего количества констант, *в разработке IaC отдают приоритет константам*. Старайтесь не назначать слишком много переменных, поскольку они усложняют обслуживание конфигурации. Вместо этого вы можете установить константу, задав в качестве значения статическую конфигурацию.

Например, Terraform использует локальные значения (www.terraform.io/docs/language/values/locals.html) для хранения констант. К традиционно определяемым константам относятся названия операционных систем, теги, идентификаторы учетных записей или имена доменов. Стандартизированные значения поставщиков инфраструктуры, таких как `internal` или `external`, для описания типа сети также могут быть константами.

2.3.5 Параметризация зависимостей

Допустим, вы создаете сервер. Вам необходимо указать сеть, которую он должен использовать. Сначала вы жестко прописываете в коде имя нужной сети – скажем, `development`. Читая конфигурацию, вы точно знаете, какую сеть использует сервер.

Но при переносе конфигурации в производство вам нужно найти и заменить каждое вхождение `development` на `production`. Проблема в том, что у вас в коде много упоминаний `development`! Процедура поиска и замены займет несколько утомительных часов.

Пример кода

Вы решили параметризовать сеть GCP как переменную, чтобы можно было воспроизвести сервер в другой среде с другой сетью. Когда вы передаете имя сети в качестве переменной, вы меняете сеть для любого сервера, ссылающегося на нее. Давайте передадим имя сети в виде переменной в коде следующим образом.

Листинг 2.1 Передача имени сети через переменную

```
import json

def hello_server(name, network): ❶
    return {
        'resource': [
            {
                'google_compute_instance': [ ❷
                    {
                        name: [
                            {
                                'allow_stopping_for_update': True,
                                'zone': 'us-central1-a',
                                'boot_disk': [
                                    {
                                        'initialize_params': [
                                            {
                                                'image': 'ubuntu-1804-lts'
                                            }
                                        ]
                                    }
                                ],
                                'machine_type': 'e2-micro',
                                'name': name,
                                'network_interface': [
                                    {
                                        'network': network ❸
                                    }
                                ],
                                'labels': {
                                    'name': name,
                                    'purpose': 'manning-infrastructure-as-code'
                                }
                            }
                        ]
                    }
                ]
            }
        ]
    }
```



```

        ]
    }
]
}

if __name__ == "__main__":
    config = hello_server(name='hello-world', network='default') ④

    with open('main.tf.json', 'w') as outfile: ⑤
        json.dump(config, outfile, sort_keys=True, indent=4) ⑤

```

- ① Передаем имя и сеть в качестве параметров конфигурации.
- ② Используем ресурс Terraform `google_compute_instance` для настройки сервера.
- ③ Задаем сеть с помощью переменной `network`.
- ④ Устанавливаем значение по умолчанию при запуске скрипта для названия сети – 'default'.
- ⑤ Создаем файл JSON с объектом сервера и запускаем его с помощью Terraform.

Эквиваленты AWS и Azure

В AWS вы должны указать ресурс Terraform `aws_instance` со ссылкой на сеть, которую вы хотите использовать (<http://mng.bz/z4j6>). Вы можете создать этот ресурс в виртуальном частном облаке (VPC) по умолчанию.

В Azure вам потребуется создать виртуальную сеть и подсети, а затем создать в сети ресурс Terraform `azurerm_linux_virtual_machine` (<http://mng.bz/064E>).

Зачем передавать имя и сеть в качестве переменных? Разработчикам часто приходится менять имя и сеть в зависимости от среды. Параметризация этих значений помогает обеспечить воспроизводимость и компоновку. Вы можете создавать новые ресурсы в разных сетях или несколько ресурсов в одной сети, не беспокоясь о конфликтах.

Выполнение примера

Здесь я продемонстрирую пошаговое выполнение примера, в результате которого будет создан сервер `hello-world`. В главе 1 приводится дополнительная информация об инструментах, необходимых для примеров, а в приложении А вы найдете подробные инструкции по использованию. Выполните следующие шаги.

- 1 Запустите скрипт на Python, введя в терминале команду

```
$ python main.py
```

Команда создает файл с расширением `*.tf.json`. Terraform автоматически выполнит поиск по этому расширению файла для создания ресурсов.

- 2 Проверьте, существует ли файл, получив список файлов в терминале:

```
$ ls *.tf.json
```

Вывод должен быть следующим:

```
main.tf.json
```

- 3 Авторизуйтесь в GCP через терминал:

```
$ gcloud auth login
```

- 4 Установите проект GCP, который вы хотите использовать, в качестве переменной среды CLOUDSDK_CORE_PROJECT:

```
$export CLOUDSDK_CORE_PROJECT=<ваш проект GCP>
```

- 5 Инициализируйте Terraform, чтобы получить плагин GCP в терминале:

```
$ terraform init
```

Вывод должен содержать следующие строки:

```
Initializing the backend...
```

```
Initializing provider plugins...
```

- Finding latest version of hashicorp/google...
- Installing hashicorp/google V3.58.0...
- Installed hashicorp/google V3.58.0 (signed by HashiCorp)

```
Terraform has created a lock file .terraform.lock.hcl to
```

```
➤record the provider selections it made above.
```

```
➤Include this file in your version control repository
```

```
➤so that Terraform can guarantee to make the same
```

```
➤selections by default when
```

```
➤you run "terraform init" in the future.
```

```
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running
```

```
➤"terraform plan" to see any changes that are
```

```
➤required for your infrastructure. All Terraform commands
```

```
➤should now work.
```

```
If you ever set or change modules or backend configuration
```

```
➤for Terraform, rerun this command to reinitialize
```

```
➤your working directory. If you forget, other
```

```
➤commands will detect it and remind you to do so if necessary.
```

- 6 Примените конфигурацию Terraform в терминале. Обязательно введите yes, чтобы применить изменения и создать экземпляр:

```
$ terraform apply
```


Вывод должен содержать конфигурацию и имя экземпляра сервера:

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

google_compute_instance.hello-world: Creating...
google_compute_instance.hello-world: Still creating... [10s elapsed]
google_compute_instance.hello-world: Still creating... [20s elapsed]
google_compute_instance.hello-world: Creation complete after 24s
➔ [id=projects/infrastructure-as-code-book/zones
➔ /us-central1-a/instances/hello-world]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

ПРИМЕЧАНИЕ В этой книге я не рассматриваю все нюансы Terraform. Подробную информацию о начале работы с Terraform можно найти в руководстве HashiCorp *Get Started* по адресу <https://learn.hashicorp.com/terraform>. Дополнительную документацию о том, как Terraform работает с GCP, можно найти по адресу <http://mng.bz/Kx2g>.

- 7 Вы можете проверить в консоли GCP сеть и метаданные сервера. В качестве альтернативы можно воспользоваться интерфейсом командной строки (CLI) Cloud SDK для проверки сети в терминале. Введите следующую команду, чтобы отфильтровать метаданные сервера hello-world:

```
$ gcloud compute instances list --filter="name=( 'hello-world' )" \
  --format="table(name,networkInterfaces.network)"
```

Вывод должен содержать URL сети в GCP:

```
NAME          NETWORK
hello-world   ['https://www.googleapis.com/compute/v1/projects/
➔ <your GCP project>/global/networks/default']
```

Сервер GCP использует сеть по умолчанию, которую вы передали в качестве переменной в текущем примере. Чтобы изменить сеть, достаточно обновить переменную. Ваш инструментарий IaC обнаружит изменения и создаст новый сервер.

Чтобы удалить сервер, нужно выполнить в терминале команду `Terraform destroy`. Обязательно введите `yes`, чтобы полностью удалить сервер:

```
$ terraform destroy
```

Когда вы определяете зависимости как переменные, вы создаете хоть и слабую, но связь двух ресурсов инфраструктуры. В главе 4 рас-

смаатриваются конкретные паттерны, которые можно использовать для дальнейшего разделения ресурсов инфраструктуры и зависимостей. Если возможно, вам следует избегать жестких зависимостей и передавать их в качестве параметров.

2.3.6 Хранение секретов

В IaC часто необходимо использовать секреты, такие как токены, пароли или ключи, для выполнения изменений на стороне провайдера.

ОПРЕДЕЛЕНИЕ *Секрет* – это часть конфиденциальной информации, такой как пароль, токен или ключ.

Когда вы создаете серверы в GCP, вам нужен ключ или токен учетной записи сервиса, который обеспечивает доступ к проекту и ресурсам сервера. Чтобы обеспечить возможность создания ресурсов, вам придется сохранять секреты как часть конфигурации инфраструктуры. Но хранение секретов в конфигурации грозит потенциальными проблемами. Если кто-то сможет прочесть ваш секрет, он сможет использовать его для доступа к вашей учетной записи GCP с возможностью создания ресурсов и получит доступ к закрытым данным!

Вам также может потребоваться передать секреты как часть конфигурации. Например, вы используете IaC для установки сертификата SSL балансировщика нагрузки. Срок действия сертификата истекает через два года. Вы воссоздаете среду два года спустя. Однако вы обнаруживаете, что срок действия зашифрованной строки сертификата истек. Вы не можете ее расшифровать и теперь должны выпустить новый сертификат.

На рис. 2.10 показано, как лучше всего защитить ваш сертификат и обеспечить возможность обновления в будущем. Вы передаете сертификат в качестве входной переменной, чтобы иметь разные сертификаты для каждой среды. Затем вы помещаете новые сертификаты в менеджер секретов, который хранит сертификаты и управляет ими.

При каждом изменении сертификата вы обновляете его в менеджере секретов. Ваш IaC обновляет свою конфигурацию при чтении сертификата из менеджера секретов. Отделение задачи управления сертификатами от настройки снижает вероятность возникновения проблем с истечением срока действия сертификата в дальнейшем.

Зачем хранить секреты за пределами IaC? Вы только что применили принципы композиемости и развиваемости для отделения секретов от других ресурсов инфраструктуры. Такое разделение гарантирует, что посторонние люди не смогут извлечь пароль или имя пользователя из файла конфигурации IaC. Вы также минимизируете влияние сбоя при ротации секрета путем повторного запуска IaC.

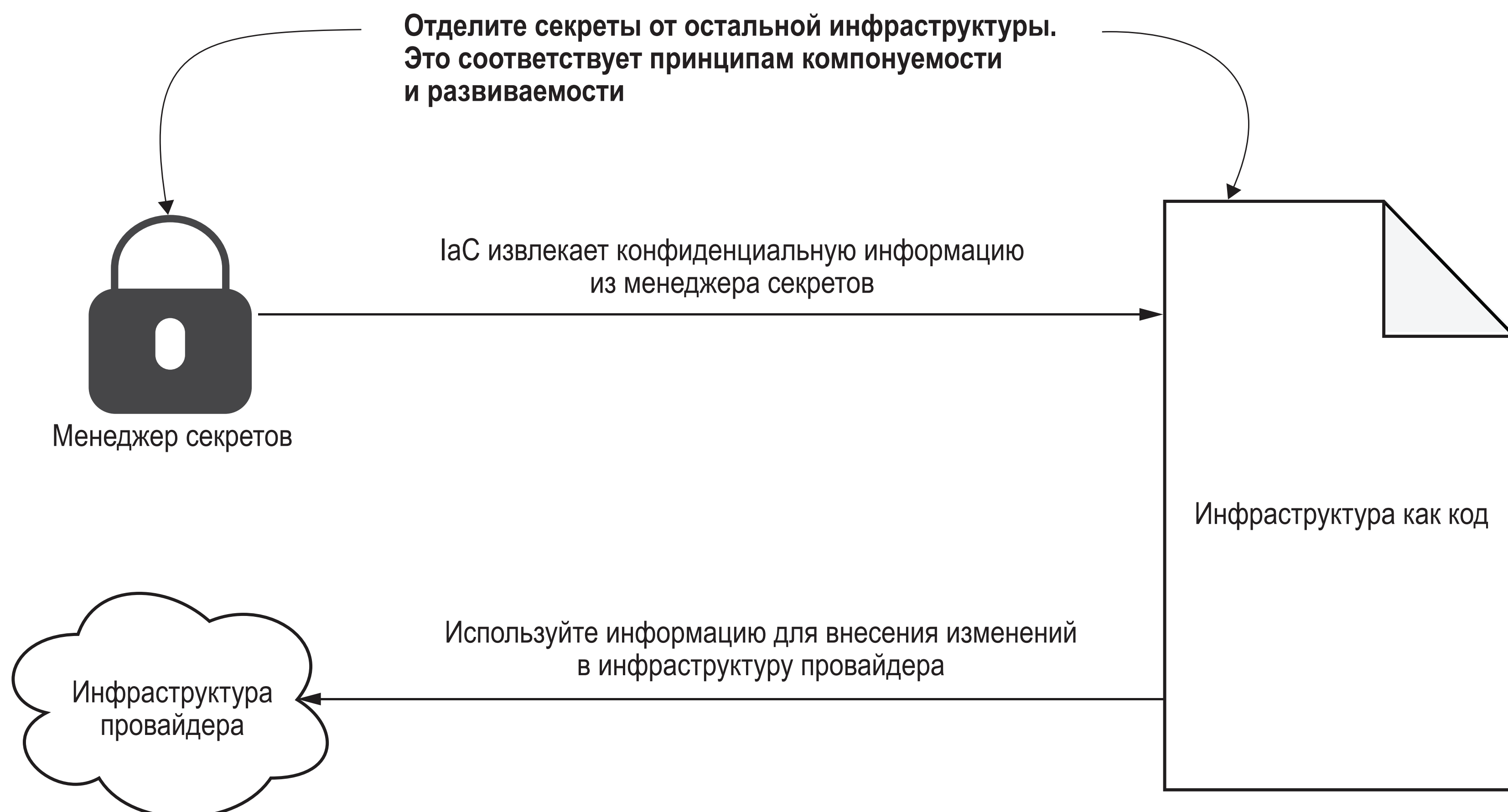


Рис. 2.10 Получение конфиденциальной информации от менеджера секретов для изменения ресурсов у провайдера инфраструктуры

Всегда передавайте секреты в IaC через переменные и работайте с ними в оперативной памяти. К таким секретам относятся ключи протокола Secure Shell (SSH), сертификаты, закрытые ключи, токены API, пароли и другая информация для входа. Отдельный объект, такой как менеджер секретов, должен хранить и управлять конфиденциальными данными аутентификации. Раздельное управление секретами облегчает воспроизведение инфраструктуры, особенно если вам нужны разные пароли и токены для каждой среды. Не следует вписывать секреты в код или передавать их в систему контроля версий в виде открытого текста.

Краткое содержание главы

- Следование принципу неизменяемости уменьшает дрейф конфигурации, сохраняет эталонное состояние инфраструктуры и улучшает ее воспроизводимость.
- В соответствии с принципом неизменяемости вместо локального изменения ресурса создают совершенно новый ресурс и обновляют состояние системы.
- Если вы вносите локальные обновления, вам необходимо согласовать изменения состояния инфраструктуры с вашей конфигурацией.
- При написании IaC используйте сообщения коммита в системе контроля версий, чтобы уведомить об изменениях и контексте, а также форматируйте код для удобства чтения.

- Параметризируйте имена, среды и зависимости от другой инфраструктуры. Если вы привязываете атрибуты конфигурации к ресурсу, то можете задать его как константу.
- Секреты всегда следует передавать как переменные, а не вписывать в код или передавать в систему контроля версий в виде открытого текста.
- При написании сценариев всегда упрощайте действия команд создания, чтения, обновления и удаления для облегчения воспроизводимости ресурсов.

Паттерны построения модулей инфраструктуры

Основные темы главы:

- группировка ресурсов инфраструктуры во взаимно komponуемые модули на основе функций;
- создание модулей инфраструктуры с использованием типовых приемов разработки ПО;
- применение паттернов модулей в типовых сценариях использования инфраструктуры.

В предыдущей главе я рассказала о фундаментальных принципах построения инфраструктуры как кода. Несмотря на знание этих принципов, мой первый скрипт автоматизации на Python представлял собой беспорядочную свалку функций в одном файле. Спустя несколько лет я изучила паттерны проектирования ПО и со временем накопила собственный набор паттернов, благодаря которому мне стало проще изменять сценарий и передавать его на обслуживание другому члену команды.

В следующих двух главах я покажу, как применять паттерны к конфигурации и зависимостям IaC. Паттерны проектирования ПО помогают выявлять распространенные проблемы и создавать повторно используемые объектно ориентированные решения.

ОПРЕДЕЛЕНИЕ Паттерн проектирования (design pattern) – это повторяемое решение распространенной задачи при разработке ПО.

Применение паттернов проектирования в разработке IaC имеет свои подводные камни. В IaC есть объекты многократного использования (ресурсы инфраструктуры). Однако их своеобразное поведение и специфика языка не позволяют напрямую позаимствовать стандартные паттерны проектирования ПО.

IaC предлагает неизменный уровень абстракции, поэтому в этой главе для аппроксимации инфраструктуры используются как *порождающие* (creational, предназначенные для создания объектов), так и *структурные* (structural, предназначенные для структурирования объектов) паттерны проектирования. При разработке сценариев IaC мы стараемся придерживаться принципа неизменности, который требует автоматически создавать новый ресурс при обновлениях. В результате паттерны проектирования, основанные на изменяемости объектов, неприменимы.

ПРИМЕЧАНИЕ Я позаимствовала и адаптировала многие паттерны из книги «Паттерны объектно ориентированного программирования» Эриха Гаммы и др. (рус. перев. Питер, 2024). Если вы хотите узнать больше об исходных паттернах проектирования программного обеспечения, я рекомендую обратиться к этой книге.

Глава содержит примеры кода Python, которые создают файлы Terraform JSON. Они ссылаются на ресурсы GCP. Вы можете перенести паттерны на выбранный вами предметно-ориентированный язык (domain-specific language, DSL), например Terraform, CloudFormation или Bicep. В зависимости от выбранного вами DSL и инструмента могут использоваться разные механизмы или функции. По возможности я отмечаю ограничения DSL и их эквивалентов для AWS и Azure.

3.1 Паттерн «одиночка»

Представьте, что вам нужно создать набор серверов баз данных в GCP с нуля. Для системы баз данных необходимы проект GCP, пользовательская сеть базы данных, шаблон конфигурирования сервера и серверной группы. Шаблон сервера устанавливает пакеты на каждый сервер, а серверная группа описывает необходимое количество серверов баз данных.

На рис. 3.1 показано, как добавить проект, сеть базы данных, шаблон сервера и серверную группу в один каталог. Вы определяете

атрибуты имени проекта GCP и его организацию. Далее вы определяете, что сеть должна иметь имя `development-database-network` с диапазоном IP-адресов `10.0.3.0/16`. Наконец, вы заявляете, что в базе данных должно быть три сервера, использующих шаблоны для MySQL. Вы записываете все эти атрибуты в виде кода в один файл конфигурации.

В конфигурации системы баз данных используется *паттерн «одиночка»* (singleton pattern), который объявляет набор ресурсов как единственный экземпляр в системе.

ОПРЕДЕЛЕНИЕ Паттерн «одиночка» объявляет набор ресурсов как единственный экземпляр в системе. Он развертывает все ресурсы с помощью одной команды.

Почему мы называем этот паттерн *одиночным*? Вы создаете *один* файл или каталог со статической конфигурацией. Кроме того, вы определяете несколько встроенных параметров для создания всех ресурсов инфраструктуры с помощью *одной* команды. Эта конфигурация описывает ресурсы, уникальные и специфичные для среды, созданной этой конфигурацией.

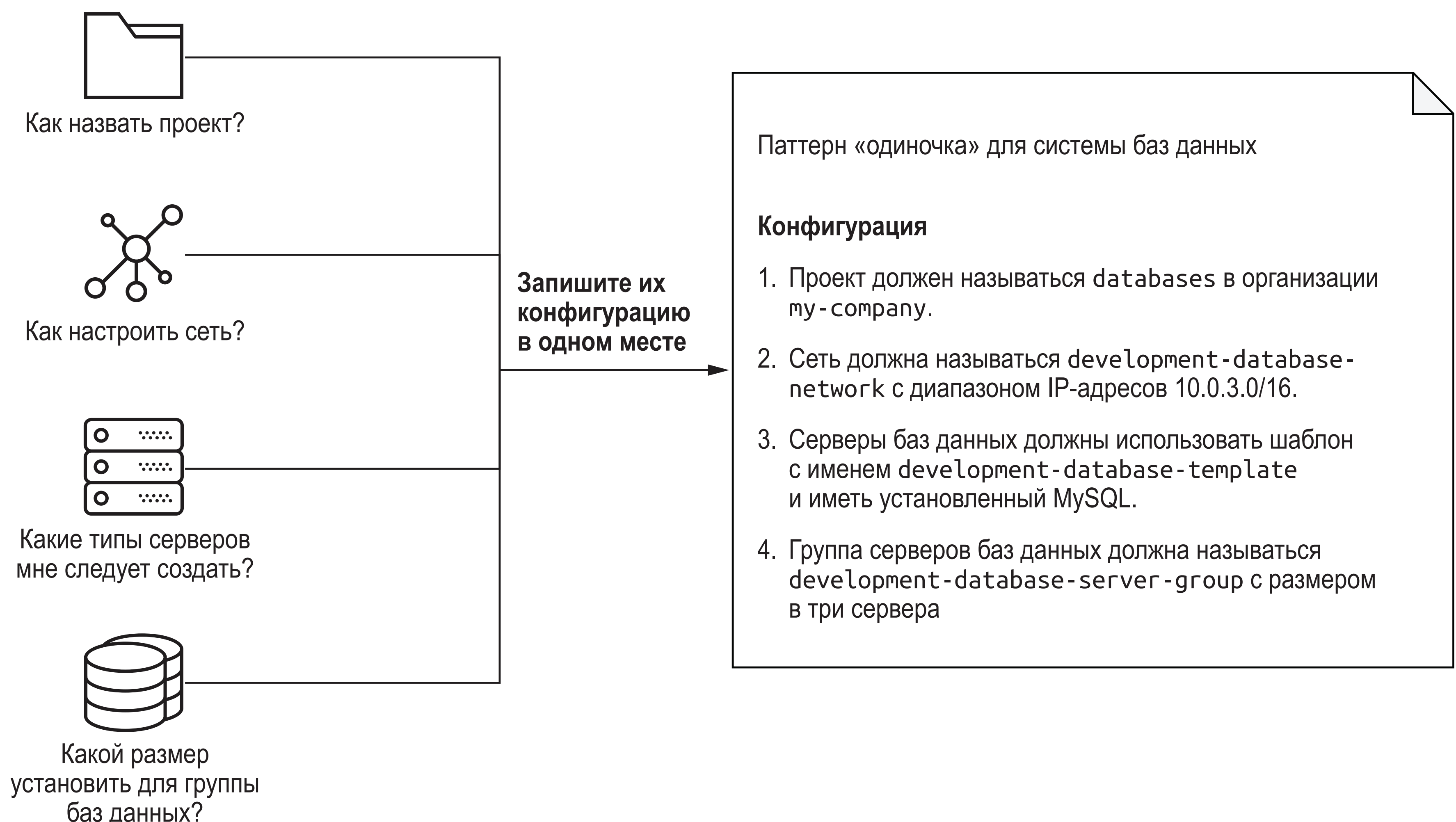


Рис. 3.1. Паттерн «одиночка» описывает конфигурацию исходного набора ресурсов, таких как проект, сеть и база данных, в одном файле. Этот паттерн описывает отношения между ресурсами в одном месте

Паттерн «одиночка» упрощает написание IaC, поскольку вы помещаете все в одну конфигурацию. Когда вы описываете каждый ресурс инфраструктуры в одной конфигурации, вы получаете единый файл кода для отладки и устранения неполадок.

Но если начать свою работу с паттерна «одиночка», это часто приводит к проблемам в дальнейшем. Когда я начала использовать паттерн «одиночка», я относилась к конфигурации инфраструктуры как к ящику для мелочей – хранилищу случайных вещей, которым в противном случае не нашлось бы места. Если не можете что-то найти, то первым делом роетесь в этом ящике (рис. 3.2).

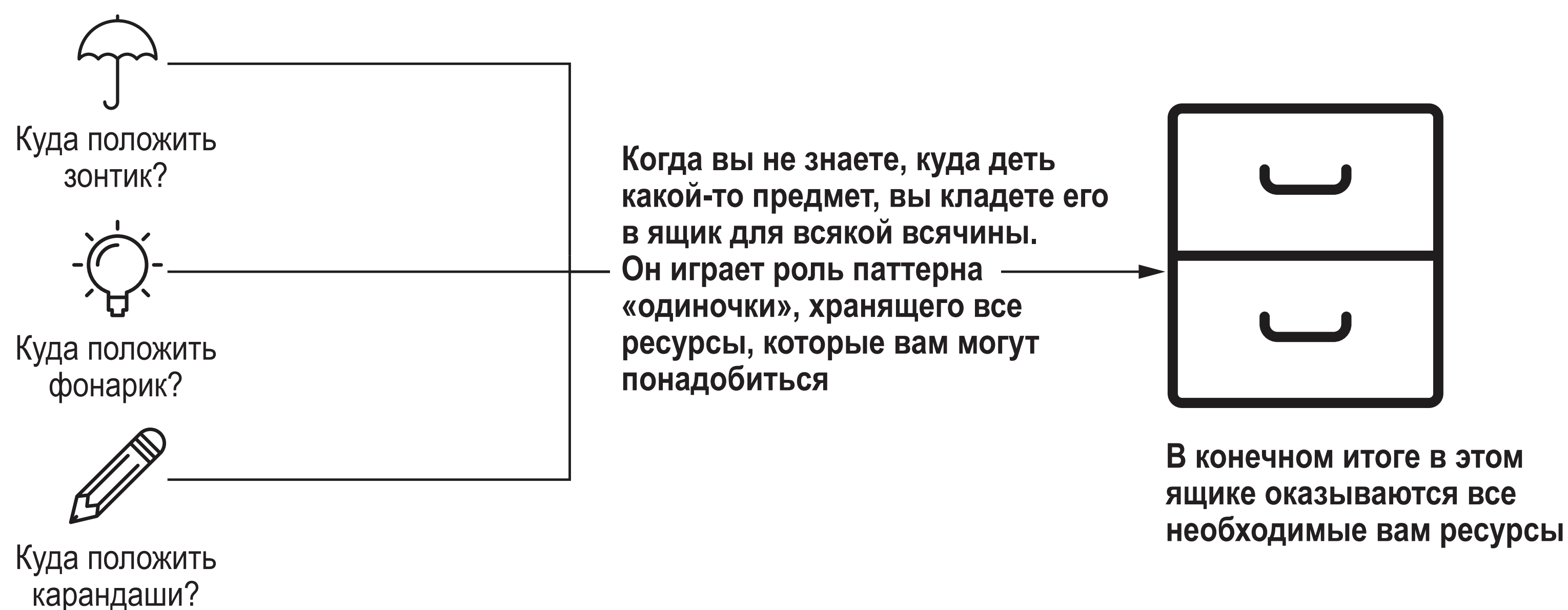


Рис. 3.2 Если вы не знаете, куда поместить объект, добавьте его в ящик для мелочей, который использует паттерн «одиночку» для агрегирования всех ресурсов

Поскольку я не знала, где еще описать ресурсы инфраструктуры, я просто добавляла их в один файл. В конце концов, этот файл превратился в беспорядочную свалку всякого хлама! Мне пришлось задуматься о паттерне «одиночка» для каждого ресурса инфраструктуры. Кроме того, большое количество ресурсов инфраструктуры в одном файле означает, что для поиска, изменения и создания ресурсов требуется время.

Паттерн «одиночка» усложняет воспроизводимость, по мере того как ваша система растет с увеличением ресурсов. Создание конфигурации для промышленной среды означает ее копирование и вставку в новый файл. Копирование и вставка конфигурации не масштабируются, если у вас появилось больше ресурсов для обновления! За наличие паттерна «одиночка» приходится платить масштабируемостью и компонуемостью. Он применим для некоторых ресурсов инфраструктуры, но не масштабируется в сложных системах.

Когда следует использовать паттерн «одиночка»? Лучше всего этот подход работает, когда у вас есть ресурс, которому требуется только один экземпляр и который редко меняется, например проект GCP. Сеть, шаблон сервера базы данных и серверная группа должны перейти в другую конфигурацию.

Все проекты GCP должны иметь уникальный идентификатор, что делает их идеальными для паттерна «одиночка». Проект может иметь только один экземпляр. Например, вы можете создать проект

с именем `databases` и сгенерировать уникальный идентификатор на основе текущего имени пользователя в системе. В следующем листинге показан код реализации паттерна «одиночка» для создания проекта GCP с именем пользователя вашей системы.

Листинг 3.1 Создание проекта в GCP с использованием паттерна «одиночка»

```
import json
import os

class DatabaseGoogleProject:
    def __init__(self):
        self.name = 'databases'
        self.organization = os.environ.get('USER')
        self.project_id = f'{self.name}-{self.organization}'
        self.resource = self._build()

    def _build(self):
        return {
            'resource': [
                {
                    'google_project': [
                        {
                            'databases': [
                                {
                                    'name': self.name,
                                    'project_id': self.project_id
                                }
                            ]
                        }
                    ]
                }
            ]
        }

if __name__ == "__main__":
    project = DatabaseGoogleProject()

    with open('main.tf.json', 'w') as outfile:
        json.dump(project.resource, outfile, sort_keys=True, indent=4)
```

- ❶ Создаем объект для базы данных проекта Google.
- ❷ Настраиваем проект Google с использованием ресурса Terraform с именем `databases`.
- ❸ Получаем пользователя операционной системы и присваиваем его переменной `organization`.
- ❹ Создаем уникальный идентификатор проекта на основе имени проекта и пользователя операционной системы, чтобы GCP мог создать проект.
- ❺ Создаем `DatabaseGoogleProject` для генерации конфигурации JSON проекта.
- ❻ Записываем словарь Python в файл JSON, который Terraform выполнит позже.

Эквиваленты AWS и Azure

Вы можете адаптировать проект GCP к аккаунту AWS. Чтобы автоматизировать создание учетных записей AWS, вам необходимо использовать AWS Organizations (<https://aws.amazon.com/organizations/>).

В Azure вы должны создать подписку и группу ресурсов. Вы можете создать группу ресурсов, используя ресурс Terraform `azurerm_resource_group` (<http://mng.bz/1orq>).

Представьте, что вы хотите создать сервер в своем проекте базы данных. Вы можете вызвать паттерн «одиночка» `DatabaseGoogleProject` и извлечь идентификатор проекта из конфигурации JSON. Паттерн содержит уникальные ресурсы, на которые можно ссылаться при вызове модуля. Например, если вы ссылаетесь на проект базы данных, вы всегда получите нужный проект, а не какой-то другой.

С вашей стороны будет разумно использовать паттерн «одиночка» для проекта GCP, потому что вы создаете его один раз и меняете нечасто. Вы можете применить паттерн «одиночка» к редко изменяющимся глобальным ресурсам, таким как учетные записи провайдеров, проекты, регистры доменных имен или корневые сертификаты SSL. Он также подходит для статических сред, таких как центры обработки данных с низким уровнем использования.

3.2 Паттерн «компоновщик»

Вместо того чтобы описывать систему баз данных с помощью паттерна «одиночка», вы можете объединить компоненты в модули. *Модуль объединяет в группу ресурсы инфраструктуры, которые относятся к одной функции или бизнес-области.* Модули позволяют изменять автоматизацию частей, не затрагивая целое.

ОПРЕДЕЛЕНИЕ Модуль группирует ресурсы инфраструктуры по функциям или бизнес-областям. В других инструментах их могут называть *инфраструктурными стеками*, или *наборами*.

Модули применяются в качестве компонентов для построения системы. Другие команды могут использовать ваши модули в качестве компонентов для своей уникальной инфраструктурной системы.

На рис. 3.3 показано, как ваша компания может быть разбита на группы, образующие структуру подчиненности. Каждая команда или менеджер подчиняется вышестоящему менеджеру, и так до уровня высшего руководства. Работа всех команд в компании подчиняется достижению общей цели.

Зачем вашей компании разбивать структуру на модули? Этот подход гарантирует, что для любой новой возможности или бизнес-инициативы найдется конкретная команда поддержки. В свою очередь, это обеспечивает компонуемость и развиваемость инфраструктуры по мере роста компании.

Ваша компания использует компонуемость для формирования команд и иерархии подотчетности

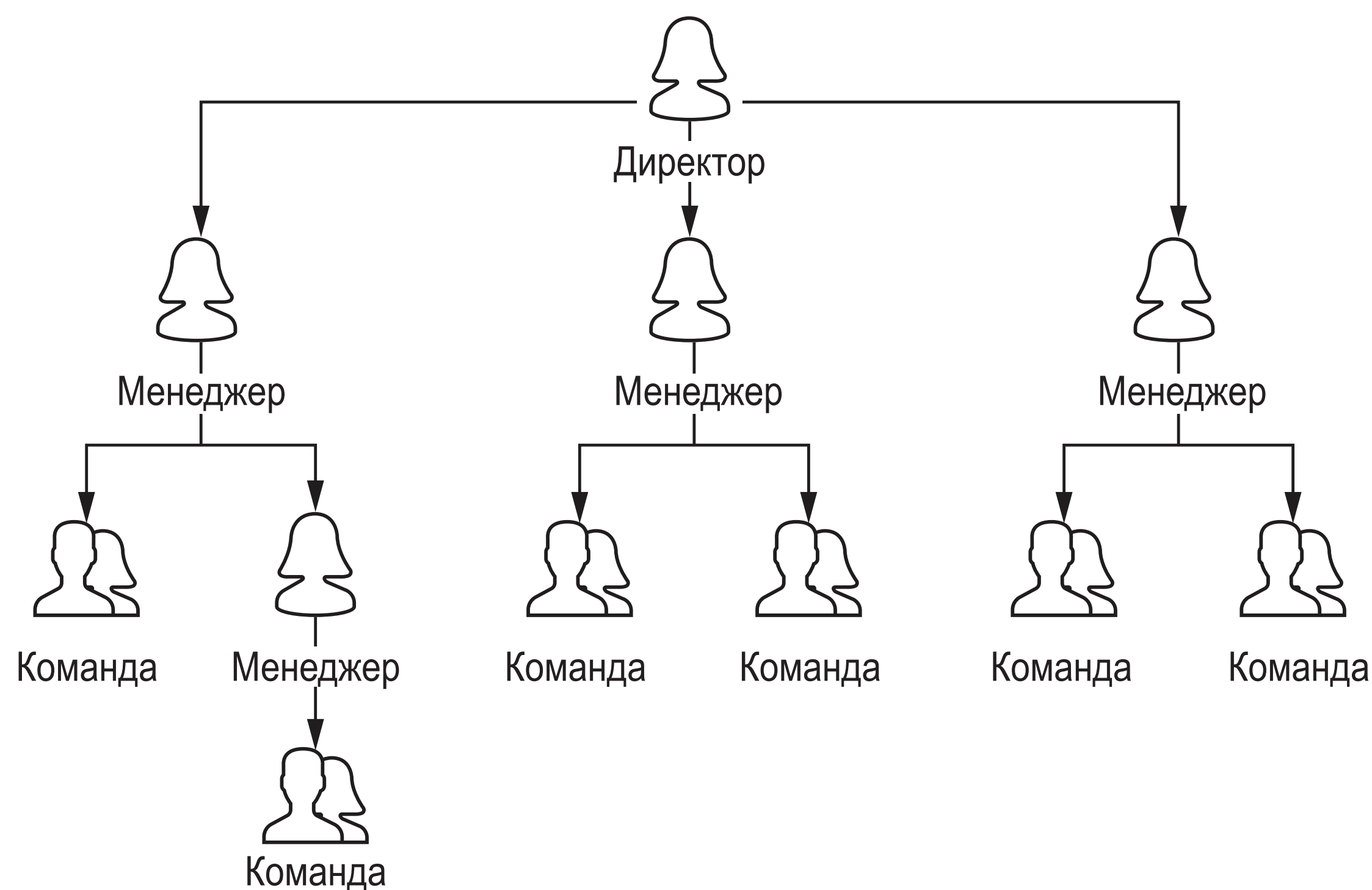


Рис. 3.3 Ваша компания может использовать паттерн «компоновщик» для группировки сотрудников в структуру подчиненности, что позволит менеджерам управлять командами и их целями

Аналогичным образом большая часть IaC зависит от *паттерна «компоновщик»* (composite pattern), предназначенного для группировки, ранжирования и структурирования набора модулей. Паттерн «компоновщик» часто рассматривают как структурный паттерн, поскольку он структурирует объекты в иерархии.

ОПРЕДЕЛЕНИЕ Паттерн «компоновщик» рассматривает модуль инфраструктуры как отдельный экземпляр и позволяет собирать, группировать и ранжировать модули в системе.

Инструменты обычно имеют собственную функцию модульности. Terraform и Vicer используют собственные фреймворки для вложения и организации модулей. Вы можете воспользоваться вложенными стеками или StackSets в CloudFormation для повторного использования шаблонов (модулей) или создания стеков в разных регионах. Инструмент управления конфигурацией, такой как Ansible, позволяет создавать наборы сценариев (playbooks) верхнего уровня, которые импортируют другие задачи.

Как реализовать модуль? Представьте, что вам нужно настроить сеть для серверов баз данных. Однако серверу нужна подсеть. Вы объединяете сеть и подсеть в модуль, как показано на рис. 3.4. Сначала вы определяете настройку сети и записываете ее в свой модуль.

Затем вы записываете конфигурацию подсети и добавляете ее в этот же модуль.

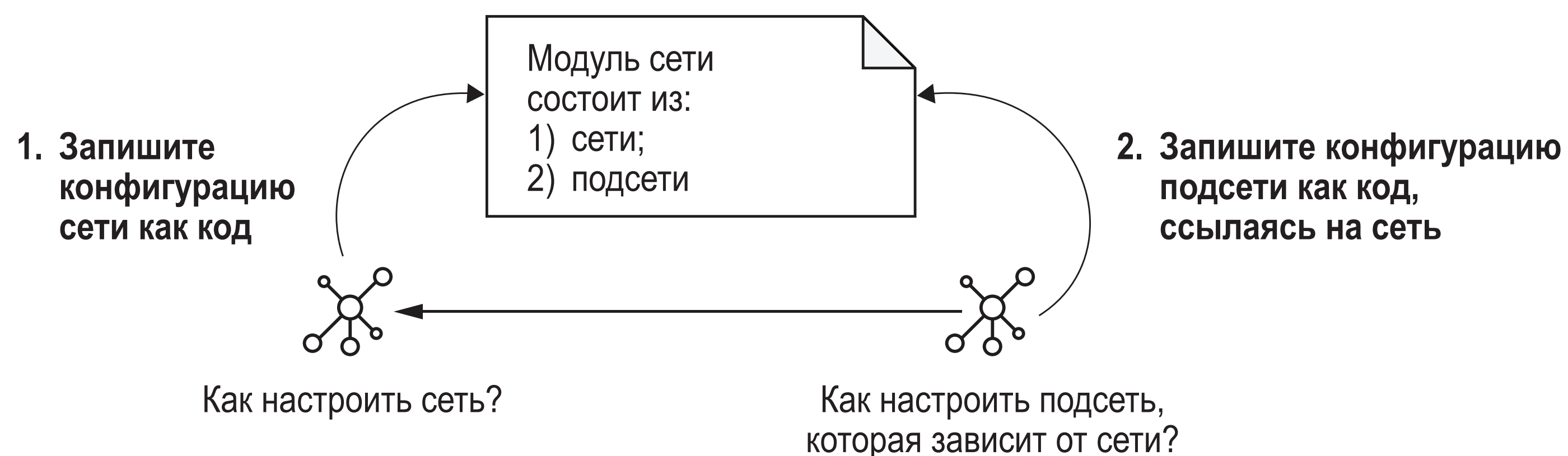


Рис. 3.4 Сетевой модуль может использовать паттерн «компоновщик» для группировки ресурсов сетей и подсетей

Теперь модуль содержит конфигурацию как для сети, так и для подсети. Если вам нужно воспроизвести сетевую систему в вашей промышленной среде, вы всегда можете скопировать и вставить весь модуль, чтобы создать новую сеть и подсеть. Составной паттерн сети гарантирует, что вы всегда сможете воспроизвести группу ресурсов, которые зависят друг от друга.

Вы можете реализовать паттерн «компоновщик» для конфигурации сети в виде модуля. Как показано в листинге 3.2, модуль создает сеть и подсеть. Вы передаете диапазоны CIDR для сети и подсети, и модуль генерирует стандартизированное имя сети.

Листинг 3.2 Создание сети и подсети

```
import json

class Network:
    def __init__(self, region='us-central1'):
        self._network_name = 'my-network'
        self._subnet_name = f'{self._network_name}-subnet'
        self._subnet_cidr = '10.0.0.0/28'
        self._region = region
        self.resource = self._build()

    def _build(self):
        return {
            'resource': [
                {
                    'google_compute_network': [
                        {
                            f'{self._network_name}': [
                                {
                                    'name': self._network_name
                                }
                            ]
                        }
                    ]
                }
            ]
        }
```



```

        },
        {
            'google_compute_subnetwork': [
                {
                    f'{self._subnet_name}': [
                        {
                            'name': self._subnet_name,
                            'ip_cidr_range': self._subnet_cidr,
                            'region': self._region,
                            'network': f'${google_compute_network
➔.{self._network_name}.name}'
                        }
                    ]
                }
            ]
        }
    ]
}

if __name__ == "__main__":
    network = Network()

    with open(f'main.tf.json', 'w') as outfile:
        json.dump(network.resource, outfile, sort_keys=True, indent=4)

```

- ❶ Создаем модуль для сети, который использует паттерн «компоновщик» для объединения сети и подсети в группу.
- ❷ Устанавливаем регион по умолчанию us-central1.
- ❸ Настраиваем сеть Google с помощью ресурса Terraform с именем my-network. GCP не требует определения сетевого блока CIDR.
- ❹ Настраиваем подсеть Google с помощью ресурса Terraform под именем my-network-subnet.
- ❺ Устанавливаем блок CIDR подсети равным 10.0.0.0/28.
- ❻ Применяем модуль для создания конфигурации JSON для сети и подсети.
- ❼ Создаем подсеть Google в сети с помощью переменной Terraform. Terraform динамически ссылается на идентификатор сети и вставляет его в конфигурацию подсети за вас.
- ❽ Настраиваем подсеть Google, используя ресурс Terraform с именем my-network-subnet.
- ❾ Устанавливаем блок CIDR подсети равным 10.0.0.0/28.
- ❿ Применяем модуль для создания конфигурации JSON для сети и подсети.
- ⓫ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Вы можете адаптировать сеть и подсети GCP к AWS VPC и подсетям или к виртуальной сети и подсетям Azure. Однако в AWS и Azure необходимо определить шлюзы и таблицы маршрутизации в каждой подсети. GCP автоматически определяет их для вас при создании сети.

Зачем создавать модуль, содержащий сразу сеть и подсеть? Вы не сможете использовать сеть GCP, если у вас нет подсети! Компоновка

позволяет создавать набор объединенных ресурсов. Объединение необходимых ресурсов в одном модуле поможет вашему товарищу по команде, который, возможно, мало что знает о сетевых технологиях. Паттерн «компоновщик» развивает принцип компоуемости, поскольку он группирует и организует общие ресурсы, которые необходимо развернуть как единое целое.

Паттерн «компоновщик» хорошо подходит для инфраструктуры, так как ресурсы инфраструктуры имеют иерархию. Модуль, следующий этому паттерну, отражает отношения между ресурсами и облегчает управление ими. Если вам необходимо обновить маршрутизацию, вы обновляете компонуемую конфигурацию сети. Вы можете обратиться к конфигурации сети, чтобы определить диапазоны CIDR подсети и вычислить адресное пространство.

Итак, вы применяете паттерн «компоновщик», группируя ресурсы по функциям, бизнес-областям или ответственности за эксплуатацию. При разработке первоначального модуля вы можете добавлять переменные, чтобы обеспечить большую гибкость благодаря параметрам или распространить свою конфигурацию среди других команд. О совместном использовании модулей я расскажу в главе 5. Однако, помимо общего паттерна «компоновщик», вы можете применять и другие паттерны, чтобы еще больше улучшить воспроизводимость IaC.

3.3 Фабричный паттерн

Ранее вы использовали паттерн «одиночка» при создании проекта GCP для системы базы данных. Затем вы применили паттерн «компоновщик» и построили сеть в другом модуле. Допустим, теперь вам понадобилась сеть базы данных, разделенная на три подсети. Вместо копирования и вставки трех подсетей вы хотите создать конфигурацию, которая принимает входные данные с именем подсети и диапазоном IP-адресов.

Конфигурация для создания трех подсетей и сети требует множества параметров, перебирать и настраивать которые может быть утомительно. Что, если бы у вас было что-то вроде фабрики по производству всех ресурсов с заданными настройками по умолчанию? На рис. 3.5 показано, что вы можете создать сетевую фабрику по производству одинаковых сетей. Вы можете сократить перечень необходимых параметров до двух входных значений, а остальные конфигурации – до значений по умолчанию.

Зная, что сеть имеет определенные атрибуты по умолчанию, вы можете минимизировать входные данные и создавать несколько ресурсов с меньшими усилиями. Я называю этот подход *фабричным паттерном* (factory pattern). Модуль, использующий фабричный паттерн, принимает набор входных данных, таких как имя и диапазон IP-адресов, и создает набор ресурсов инфраструктуры.

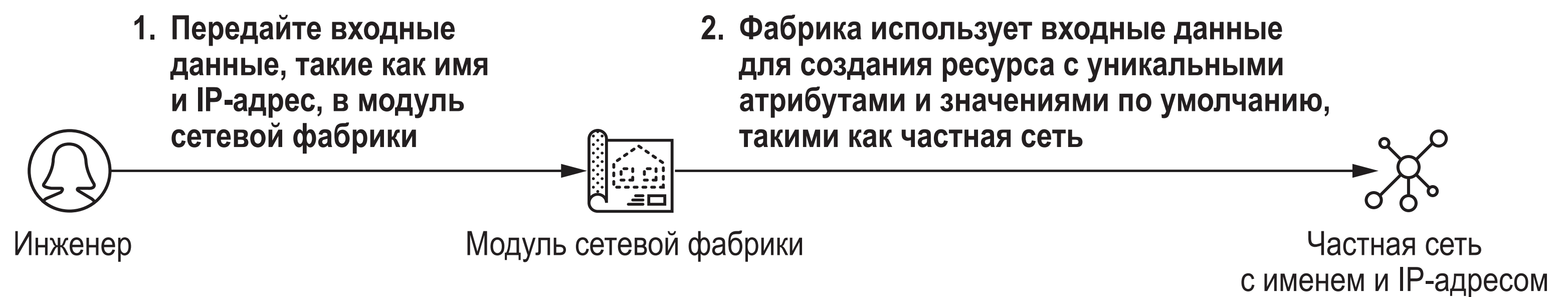


Рис. 3.5 Модуль фабричного паттерна содержит набор значений по умолчанию для минимальной конфигурации ресурсов и позволяет настраивать ее, принимая входные переменные

ОПРЕДЕЛЕНИЕ *Фабричный паттерн* принимает набор входных переменных и создает набор ресурсов инфраструктуры на основе входных переменных и констант по умолчанию.

Вам хотелось бы сохранить достаточную гибкость для внесения изменений, таких как IP-адрес и имя подсети. В общем, вам необходимо найти в модуле баланс между гибкостью настройки и использованием атрибутов по умолчанию. В конце концов, вы хотите придерживаться принципа воспроизводимости, но сохранять возможность развития ресурса. Мы подробнее обсудим совместное использование ресурсов, таких как модули, в главе 5, а также стандартизацию методов обеспечения безопасности модулей в главе 8.

Вернемся к нашему примеру. Как создать три подсети без передачи списка имен? Модуль может автоматически присваивать имена подсетям, чтобы не пришлось прописывать их в коде. На рис. 3.6 показано, как добавить логику в модуль сетевой фабрики для генерации стандартного имени подсети на основе сетевого адреса.

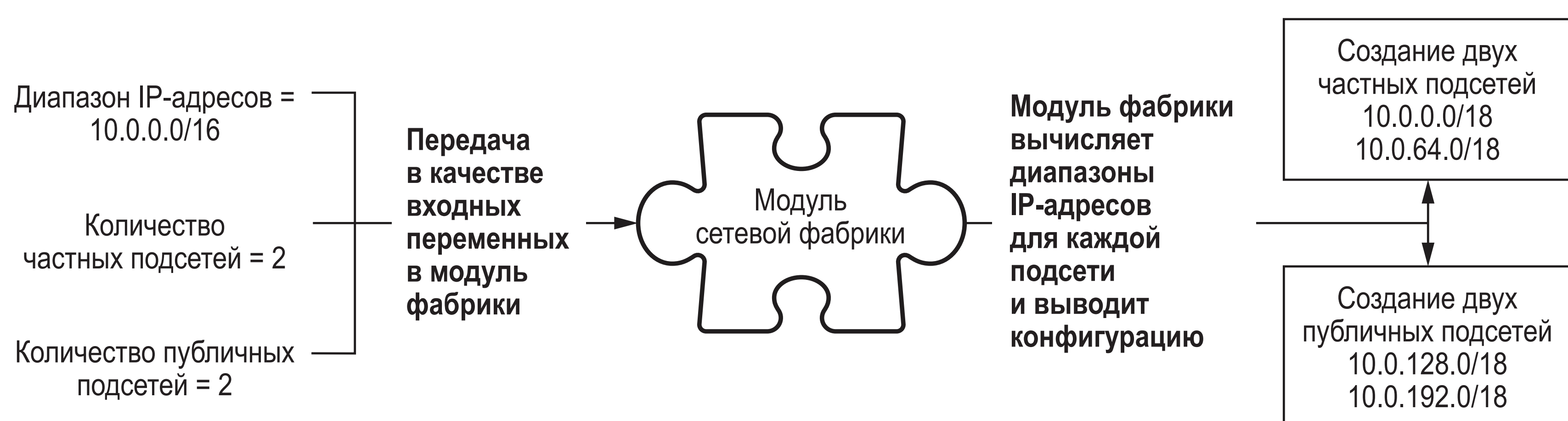


Рис. 3.6 Модуль сетевой фабрики может содержать преобразования, которые вычисляют адресацию подсети и создают несколько ресурсов подсети для сети

Модули, использующие фабричный паттерн, могут преобразовывать входные переменные в стандартизированные шаблоны, что является обычной практикой при создании имен или идентификаторов. Когда вы реализуете сетевой модуль с фабричным паттерном в коде, вы добавляете модуль `SubnetFactory`. В листинге 3.3 создается модуль фабрики для генерации имен подсетей.

Листинг 3.3 Создание трех подсетей в GCP с использованием фабричного паттерна

```

import json
import ipaddress

def _generate_subnet_name(address):
    address_identifier = format(ipaddress.ip_network(
        address).network_address).replace('.', '-')
    return f'network-{address_identifier}'

class SubnetFactory:
    def __init__(self, address, region):
        self.name = _generate_subnet_name(address)
        self.address = address
        self.region = region
        self.network = 'default'
        self.resource = self._build()

    def _build(self):
        return {
            'resource': [
                {
                    'google_compute_subnetwork': [
                        {
                            f'{self.name}': [
                                {
                                    'name': self.name,
                                    'ip_cidr_range': self.address,
                                    'region': self.region,
                                    'network': self.network
                                }
                            ]
                        }
                    ]
                }
            ]
        }

if __name__ == "__main__":
    subnets_and_regions = {
        '10.0.0.0/24': 'us-central1',
        '10.0.1.0/24': 'us-west1',
        '10.0.2.0/24': 'us-east1',
    }

    for address, region in subnets_and_regions.items():
        subnetwork = SubnetFactory(address, region)

        with open(f'{_generate_subnet_name(address)}.tf.json',
            'w') as outfile:
            json.dump(subnetwork.resource, outfile,
                sort_keys=True, indent=4)

```


- ❶ Генерируем имя для заданной подсети, разделяя диапазон IP-адресов дефисом и добавляя его к «network».
- ❷ Создаем модуль для подсети на основе фабричного паттерна для создания любого количества подсетей.
- ❸ Передаем адрес подсети в фабрику.
- ❹ Передаем регион подсети в фабрику.
- ❺ В этом примере создаются подсети в сети default.
- ❻ Применяем модуль для создания конфигурации JSON для сети и подсети.
- ❼ Создаем подсеть Google с использованием ресурса Terraform на основе имени, адреса, региона и сети.
- ❽ С помощью модуля фабрики создаем подсеть, определенную диапазоном IP-адресов и регионом.
- ❾ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Почему я выделяю подсети в отдельные модули фабрики? Создание отдельного модуля для подсетей способствует реализации принципа развиваемости. Я могу изменить логику генерации имен для любого количества подсетей. Я также могу обновить формат имени, не затрагивая сеть.

Большинство модулей фабрик предусматривают преобразование или динамическое создание атрибутов. Например, вы можете изменить модуль сетевой фабрики для расчета диапазонов IP-адресов подсетей. В результате расчетов автоматически создается правильное количество частных или общедоступных подсетей.

Однако я рекомендую по возможности свести к минимуму преобразования, которые вы добавляете в модули фабрик. Они могут усложнить конфигурацию ресурсов. Чем сложнее логика вашего преобразования, тем больше вам нужно тестов для проверки преобразований. Я расскажу о тестировании модулей и конфигурации инфраструктуры в главе 6.

Фабричный паттерн обеспечивает баланс между воспроизводимостью и развиваемостью ресурсов инфраструктуры. Он производит аналогичную инфраструктуру с небольшими различиями в именах, размерах или других атрибутах. Вам понадобится создать модуль фабрики, если конфигурация применима к часто создаваемым ресурсам, таким как сети или серверы.

Каждый раз, когда вы запускаете модуль фабрики, вы можете рассчитывать на получение определенного набора ресурсов, который запрашиваете. Модуль не содержит особой логики для определения того, какие ресурсы создавать. Вместо этого модуль сосредоточен на настройке атрибутов ресурсов.

Я часто пишу модули фабрик со множеством констант по умолчанию и небольшим количеством входных переменных. Таким образом я сокращаю накладные расходы на поддержание и проверку входных данных. Инфраструктура, в которой обычно используются фабрики, состоит из сети и подсетей, кластеров серверов, управляемых баз данных, управляемых очередей или управляемых кешей и многого другого.

3.4 Паттерн «прототип»

Теперь, когда у вас есть модуль для построения сетей баз данных, вы можете создавать серверы баз данных. Однако вы должны пометить все ресурсы в системе базы данных именами заказчика, бизнес-подразделения и центра учета затрат. Команда аудита также попросит вас включить в конфигурацию атрибут `automated=true` для идентификации автоматизированных ресурсов.

В идеале теги (или метки в GCP) должны быть одинаковыми для всех ваших ресурсов. При обновлении тегов ваш сценарий автоматизации должен копировать их на каждый ресурс. Вы узнаете больше о важности тегирования в главе 8.

А что, если собрать все теги в одном месте и обновить их одновременно? На рис. 3.7 показано, что вы можете поместить все свои теги в модуль. Сервер базы данных ссылается на общий модуль для тегов и применяет статические значения к серверу.

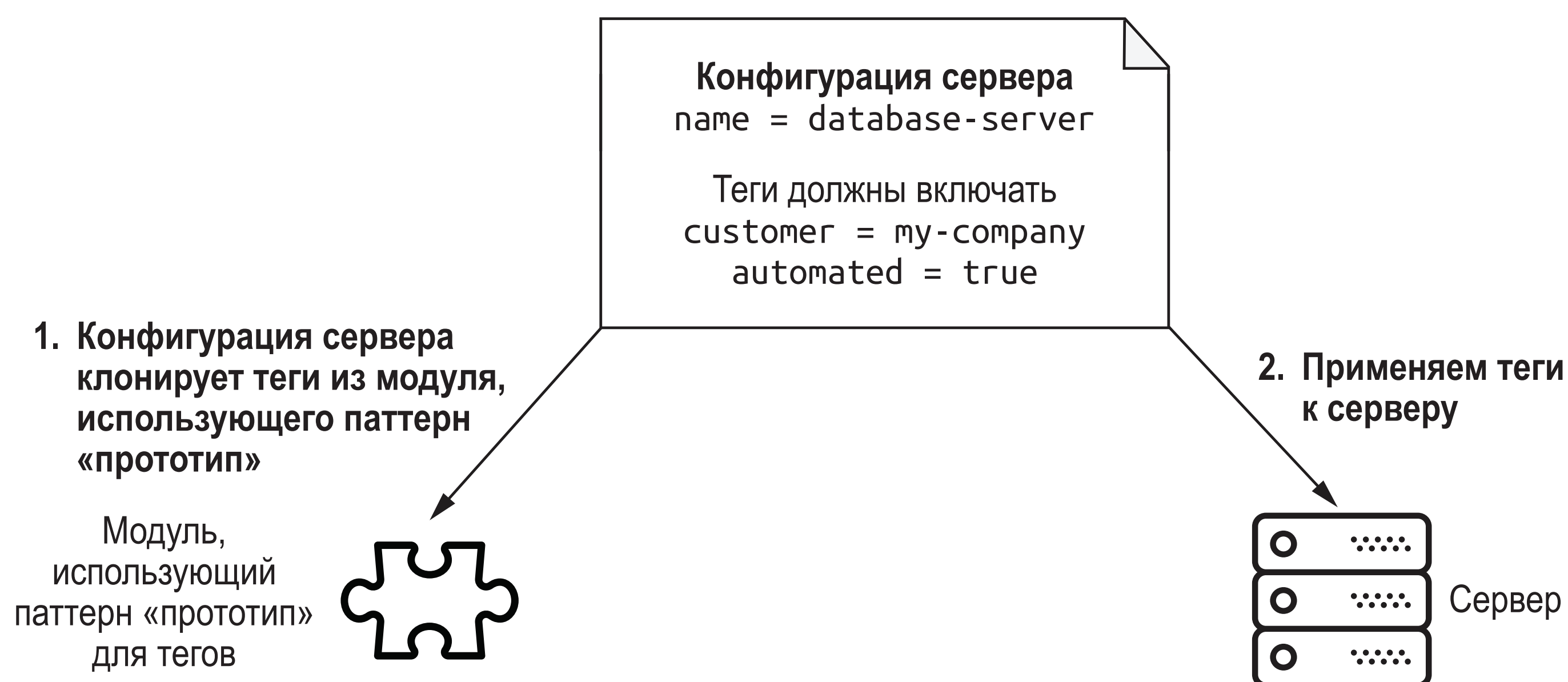


Рис. 3.7 Модуль на основе паттерна «прототип» возвращает копию статических значений, таких как теги, для использования другими ресурсами инфраструктуры

Вместо того чтобы жестко запрограммировать каждый тег, мы создали модуль, реализующий *паттерн «прототип»* (prototype pattern), чтобы выразить набор статических значений по умолчанию для использования другими модулями. Модули, использующие паттерн «прототип», создают копию конфигурации для добавления к другим ресурсам.

ОПРЕДЕЛЕНИЕ Паттерн «прототип» использует набор входных переменных для создания набора статических значений по умолчанию, получаемых другими модулями. Обычно они не создают ресурсы инфраструктуры напрямую, а экспортируют значения конфигурации.

Прототип можно рассматривать как словарь, в котором хранятся слова и определения (рис. 3.8). Создатели словаря меняют слова

и определения. Вы можете ссылаться на словарь и обновлять свой текст или словарный запас.

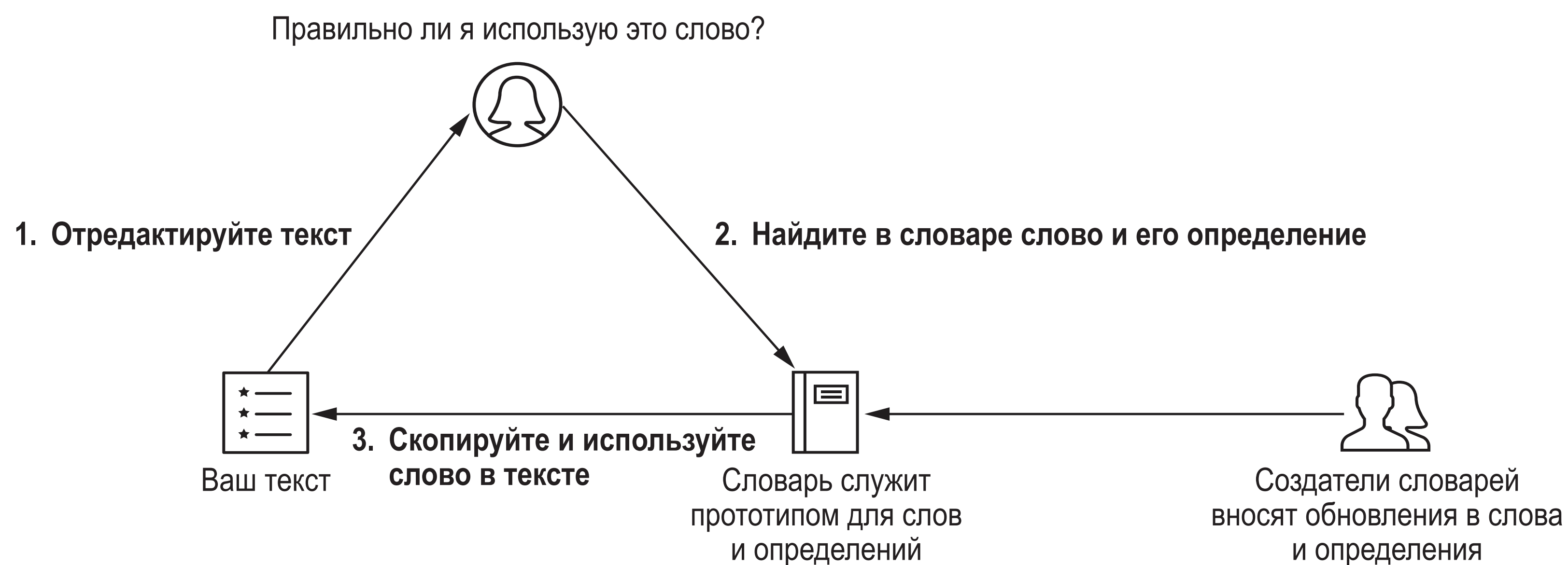


Рис. 3.8 Вы используете словарь в качестве прототипа для ссылок на слова и определения и обновляете их при написании текста

Зачем применять модуль, использующий паттерн «прототип» для ссылки на общие метаданные? Паттерн «прототип» реализует принцип развиваемости и воспроизводимости. Он обеспечивает согласованную конфигурацию всех ресурсов и упрощает развитие общей конфигурации. Вам не нужно искать и заменять строки в файлах!

Давайте реализуем модуль тегов с помощью паттерна «прототип». В листинге 3.4 с помощью такого паттерна создается модуль, который возвращает набор стандартных тегов. В последующих ресурсах инфраструктуры необходимо ссылаться на модуль `StandardTags` для всех тегов, которые вам нужно включить. Модуль не создает ресурсы тегов. Вместо этого он возвращает копию предопределенных тегов.

Листинг 3.4 Создание модуля тегов с использованием паттерна «прототип»

```
import json

class StandardTags():
    def __init__(self):
        self.resource = {
            'customer': 'my-company',
            'automated': True,
            'cost_center': 123456,
            'business_unit': 'ecommerce'
        }

class ServerFactory:
    def __init__(self, name, network, zone='us-central1-a', tags={}):
        self.name = name
        self.network = network
        self.zone = zone
        self.tags = tags
        self.resource = self._build()
```



```

def _build(self):
    return {
        'resource': [
            {
                'google_compute_instance': [ 5
                    {
                        self.name: [
                            {
                                'allow_stopping_for_update': True,
                                'boot_disk': [
                                    {
                                        'initialize_params': [
                                            {
                                                'image': 'ubuntu-1804-lts'
                                            }
                                        ]
                                    }
                                ],
                                'machine_type': 'e2-micro',
                                'name': self.name,
                                'network_interface': [
                                    {
                                        'network': self.network
                                    }
                                ],
                                'zone': self.zone,
                                'labels': self.tags 6
                            }
                        ]
                    }
                ]
            }
        ]
    }

if __name__ == "__main__":
    config = ServerFactory( 7
        name='database-server', network='default', 7
        tags=StandardTags().resource) 8

    with open('main.tf.json', 'w') as outfile: 9
        json.dump(config.resource, outfile, 9
                   sort_keys=True, indent=4) 9

```

- ❶ Создаем модуль, используя паттерн «прототип», который возвращает копию стандартных тегов, таких как заказчик, центр учета затрат и бизнес-подразделение.
- ❷ Создаем модуль, используя фабричный паттерн, чтобы сгенерировать экземпляр сервера Google на основе имени, сети и тегов.
- ❸ Передаем теги как переменную серверному модулю.
- ❹ Используем модуль для создания конфигурации JSON для сервера в сети default.
- ❺ Создаем экземпляр сервера Google с использованием ресурса Terraform.
- ❻ Добавляем теги, хранящиеся в переменной, в ресурс экземпляра сервера Google.
- ❼ Используем модуль для создания конфигурации JSON сервера в сети default.

- ⑧ Используем стандартный модуль тегов для добавления тегов на сервер.
- ⑨ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Чтобы перенести код из листинга 3.4 к другому провайдеру облачных услуг, измените ресурс на экземпляр Amazon EC2 или виртуальную машину Azure Linux. Затем передайте `self.tags` в атрибут `tags` для ресурса AWS или Azure.

Теперь запустите скрипт Python для создания конфигурации сервера, как показано в листинге 3.5. Когда вы исследуете выходные данные JSON для сервера, вы заметите, что сервер содержит набор меток. Эти метки соответствуют стандартным меткам вашего модуля, использующего паттерн «прототип»!

Листинг 3.5 Создание конфигурации сервера с тегами из модуля, использующего паттерн «прототип»

```
{
  "resource": [
    {
      "google_compute_instance": [
        {
          "database-server": [
            {
              "allow_stopping_for_update": true,
              "boot_disk": [
                {
                  "initialize_params": [
                    {
                      "image": "ubuntu-1804-lts"
                    }
                  ]
                }
              ],
              "labels": {
                "automated": true,
                "business_unit": "ecommerce",
                "cost_center": 123456,
                "customer": "my-company"
              },
              "machine_type": "e2-micro",
              "name": "database-server",
              "network_interface": [
                {
                  "network": "default"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```



```

        "zone": "us-central1-a"
    }
]
}

```

4

- ❶ Файл JSON определяет экземпляр сервера Google, использующий ресурс Terraform.
- ❷ Terraform идентифицирует ресурс как сервер базы данных. Конфигурация JSON соответствует той, которую вы определили в модуле фабрики серверов с помощью Python.
- ❸ Добавляем теги из модуля, использующего паттерн «прототип» стандартных тегов в поле меток в конфигурации сервера.
- ❹ Конфигурация JSON извлекает переменную зоны и записывает ее в файл JSON.

Обратите внимание, что конфигурация вашего сервера содержит множество жестко запрограммированных значений, таких как операционная система и тип компьютера. Значения ведут себя как глобальные настройки по умолчанию. Со временем вы продолжите добавлять глобальные настройки в свой модуль фабрики и обнаружите, что они переполняют модуль!

Чтобы распутать и упорядочить глобальные настройки по умолчанию, вы можете определить их в модуле, использующем паттерн «прототип». Модуль упрощает изменение значения по умолчанию в будущем и компоновку его с другими значениями. Прототип становится статическим, четко определенным значением по умолчанию для ресурса.

В одной из таких ситуаций я начала с написания модуля фабрики для создания набора оповещений об инфраструктуре. Сначала я передавала имена сред и пороговые значения метрик для параметризации оповещений и их конфигурации. Я обнаружила, что для оповещений не требуются имена сред, а пороговые значения метрик не меняются в разных средах. В результате я превратила этот модуль в прототип. Команды, которым нужно было добавить метрики в свои системы, импортировали модуль. Модуль добавил в их конфигурацию предустановленные ресурсы оповещений.

Предметно-ориентированные языки (DSL)

DSL для таких инструментов, как Terraform, Kubernetes, CloudFormation и Vicer, не имеют глобальных констант, как в обычных языках программирования. Однако они поддерживают ссылки на модули и структуры объектов. Вы можете использовать один и тот же паттерн для DSL и языков программирования, создав прототип как объект.

Прототип упрощает создание стандартного набора ресурсов или конфигураций. Он устраняет неопределенность при установке вход-

ных значений. Однако у вас могут быть нестандартные значения. Например, значения, основанные на самом ресурсе. Обычно я объединяю собственные теги, уникальные для ресурса, со списком стандартных тегов.

Помимо тегов, я обычно применяю модули, использующие паттерн «прототип» для регионов, зон доступности или идентификаторов учетных записей. Я создаю модули с паттерном «прототип», когда у меня есть статические конфигурации со многими глобальными настройками по умолчанию или сложными преобразованиями. Например, у вас может быть сценарий инициализации сервера, который запускается при использовании SSL. Вы можете создать модуль для шаблона сценария, который зависит от того, используете ли вы SSL.

3.5 Паттерн «строитель»

Вы научились применять паттерн «одиночка» для создания проекта, фабрику для создания сетей и прототип для установки тегов сервера базы данных. Далее вы создадите балансировщик нагрузки, который организует соединение с базой данных.

Но сначала вы столкнетесь со сложным требованием. Модуль должен позволять создавать частные или публичные балансировщики нагрузки! Частный балансировщик нагрузки требует другой конфигурации сервера и сети. Вы должны создать модуль, который обеспечивает гибкий выбор частного или публичного балансировщика нагрузки и настраивает сервер и сети в соответствии с вашим выбором.

На рис. 3.9 показан модуль, который выбирает конфигурацию брандмауэра и сервера в зависимости от типа вашего балансировщика нагрузки. Вы можете использовать один и тот же модуль для создания внешнего или внутреннего балансировщика нагрузки. Модуль обеспечивает правильное конфигурирование балансировщика нагрузки и необходимые правила брандмауэра.

Модуль дает вам возможность построить систему, которую вы хотите, обеспечивая возможность развития и компоновки. Модуль руководствуется *паттерном «строитель»* (builder pattern), который фиксирует некоторые значения по умолчанию, но позволяет вам составить нужную систему. Паттерн «строитель» организует набор связанных ресурсов, которые вы включаете или отключаете для желаемой системы.

ОПРЕДЕЛЕНИЕ Паттерн «строитель» собирает набор ресурсов инфраструктуры, которые вы можете включить или отключить для достижения желаемой конфигурации.

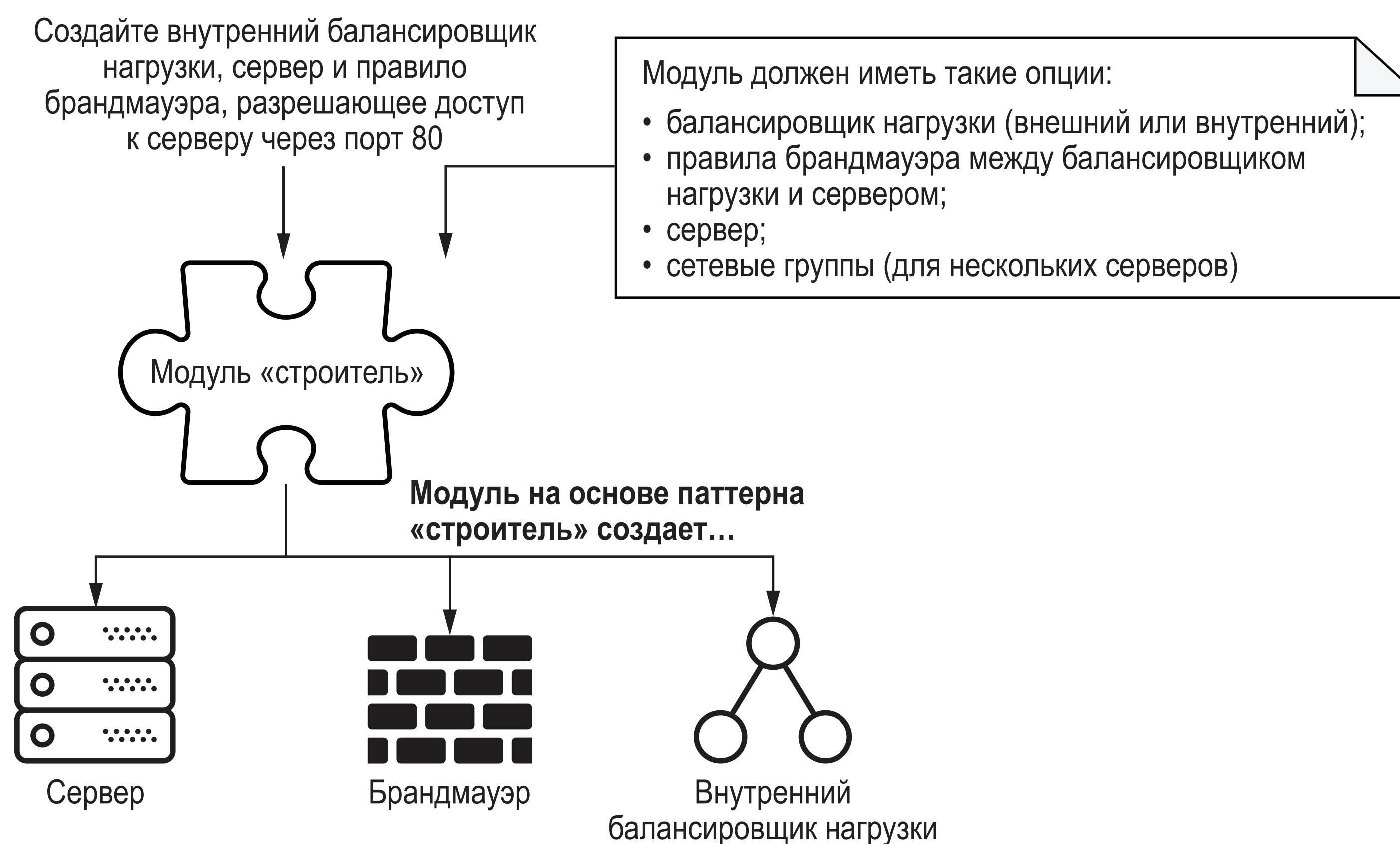


Рис. 3.9 Модуль, использующий паттерн «строитель» базы данных, должен учитывать параметры выбора типа балансировщика нагрузки и правил брандмауэра, которые модуль должен создать

Реализация паттерна «строитель» в модуле базы данных позволяет вам генерировать комбинацию ресурсов на основе вашего выбора. Паттерн «строитель» использует входные данные, чтобы *решить*, какие ресурсы ему необходимо построить, в то время как модуль фабрики послушно настраивает атрибуты ресурсов на основе входных переменных. Этот паттерн работает как строитель дома. Вы выбираете один из заранее подготовленных типовых чертежей и сообщаете строителю, какие изменения вы хотите внести в планировку (рис. 3.10). Например, некоторые строители могут по вашей просьбе добавить дополнительную комнату, убрав гараж.

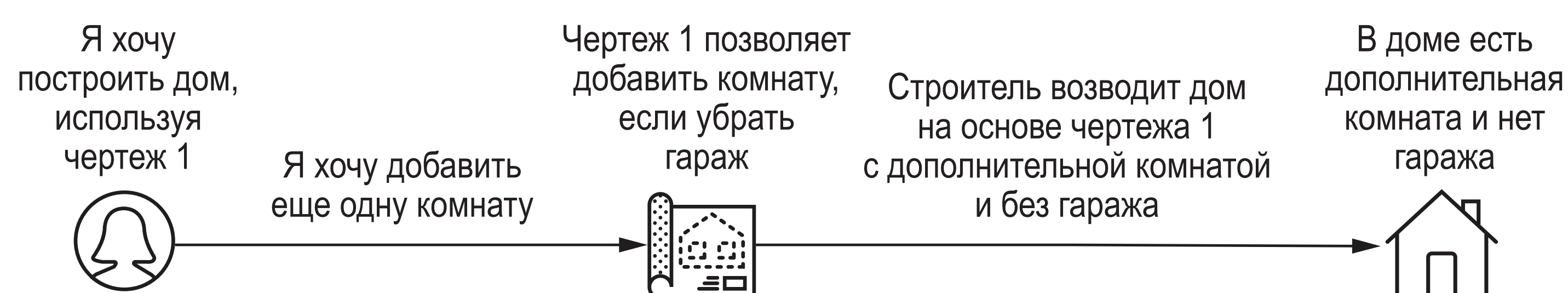


Рис. 3.10 Модуль, использующий паттерн «строитель», возводит дом по заранее заданному проекту, который позволяет вносить изменения в планировку, например добавлять дополнительную комнату

Пример реализации паттерна «строитель» показан в листинге 3.6. Сначала мы определяем модуль балансировщика нагрузки, используя фабричный паттерн. Мы используем фабричный паттерн для настройки балансировщика нагрузки (также называемого *правилом переадресации вычислений*, *Google compute forwarding* в GCP). Модуль настраивает схему балансировщика нагрузки как внешнюю или внутреннюю.

Листинг 3.6 Использование модуля фабрики для настройки балансировщика нагрузки

```

class LoadBalancerFactory: ❶
    def __init__(self, name, region='us-central1', external=False): ❶
        self.name = name
        self.region = region
        self.external = external ❶
        self.resources = self._build() ❷

    def _build(self):
        scheme = 'EXTERNAL' if self.external else 'INTERNAL' ❸
        resources = []
        resources.append({
            'google_compute_forwarding_rule': [{ ❹
                'db': [
                    {
                        'name': self.name,
                        'target': r'${google_compute_target_pool.db.id}', ❺
                        'port_range': '3306', ❻
                        'region': self.region,
                        'load_balancing_scheme': scheme,
                        'network_tier': 'STANDARD'
                    }
                ]
            }
        ]
    })
    return resources

```

- ❶ Создаем модуль для балансировщика нагрузки, который использует фабричный паттерн для создания внутреннего или внешнего балансировщика нагрузки.
- ❷ Применяем модуль для создания файла конфигурации JSON балансировщика нагрузки.
- ❸ Устанавливаем внешнюю или внутреннюю схему балансировки нагрузки. Балансировщик нагрузки по умолчанию использует внутреннюю схему.
- ❹ Создаем правило переадресации вычислений Google с использованием ресурса Terraform. Это эквивалент балансировщика нагрузки в GCP.
- ❺ Назначаем целью балансировщика нагрузки группу серверов базы данных. При этом используется встроенная функция интерполяции переменных Terraform для динамического разрешения идентификатора группы серверов базы данных.
- ❻ Разрешаем трафик на порт 3306 базы данных MySQL.

Эквиваленты AWS и Azure

Правило переадресации вычислений GCP можно преобразовать в AWS Elastic Load Balancing (ELB) или Azure Load Balancer. Аналогично группа безопасности AWS или группа безопасности сети Azure примерно соответствует правилу брандмауэра GCP. Пример использования AWS доступен в репозитории кода по адресу <https://github.com/joatmon08/manning-book>.

Нам нужно помнить, что внешний балансировщик нагрузки требует дополнительной настройки правила брандмауэра. Необходимо разрешить трафик из внешних источников на порты базы данных. Давайте определим модуль, используя фабричный паттерн для другого правила брандмауэра, чтобы разрешить трафик из внешних источников, как показано в листинге 3.7.

Листинг 3.7 Использование фабричного модуля для правила брандмауэра

```
class FirewallFactory: ❶
    def __init__(self, name, network='default'):
        self.name = name
        self.network = network
        self.resources = self._build() ❷

    def _build(self):
        resources = []
        resources.append({
            'google_compute_firewall': [{ ❸
                'db': [
                    {
                        'allow': [ ❹
                            { ❹
                                'protocol': 'tcp', ❹
                                'ports': ['3306'] ❹
                            } ❹
                        ], ❹
                        'name': self.name,
                        'network': self.network,
                        'source_ranges': ['0.0.0.0/0'] ❹
                    }
                ]
            }
        ]
        })
        return resources
```

- ❶ Создаем модуль брандмауэра, который использует фабричный паттерн для создания правила брандмауэра.
- ❷ Применяем модуль для создания файла конфигурации JSON балансировщика нагрузки.
- ❸ Создаем брандмауэр Google с использованием ресурса Terraform. Это эквивалент правила брандмауэра в GCP.
- ❹ Правило брандмауэра должно по умолчанию разрешать TCP-трафик на порт 3306.

В соответствии с принципом компоновки мы помещаем балансировщик нагрузки и модуль фабрики в модуль, использующий паттерн «строитель» базы данных. Модулю нужна переменная, которая поможет нам выбрать тип балансировщика нагрузки и указать, следует ли включать правило брандмауэра, разрешающее трафик к балансировщику нагрузки.

Когда мы реализуем модуль «строитель» базы данных в листинге 3.8, мы настраиваем его на создание группы серверов базы данных и сетей по умолчанию. Тогда модуль «строитель» принимает два варианта: внутренний или внешний балансировщик нагрузки и дополнительное правило брандмауэра.

Листинг 3.8 Создание базы данных с помощью паттерна «строитель»

```
import json
from server import DatabaseServerFactory
from loadbalancer import LoadBalancerFactory
from firewall import FirewallFactory

class DatabaseModule:
    def __init__(self, name):
        self._resources = []
        self._name = name
        self._resources = DatabaseServerFactory(self._name).resources

    def add_internal_load_balancer(self):
        self._resources.extend(
            LoadBalancerFactory(
                self._name, external=False).resources)

    def add_external_load_balancer(self):
        self._resources.extend(
            LoadBalancerFactory(
                self._name, external=True).resources)

    def add_google_firewall_rule(self):
        self._resources.extend(
            FirewallFactory(
                self._name).resources)

    def build(self):
        return {
            'resource': self._resources
        }

if __name__ == "__main__":
    database_module = DatabaseModule('development-database')
    database_module.add_external_load_balancer()
    database_module.add_google_firewall_rule()

    with open('main.tf.json', 'w') as outfile:
        json.dump(database_module.build(), outfile,
                  sort_keys=True, indent=4)
```

- ❶ Импортируем модули для создания группы серверов базы данных, балансировщика нагрузки и брандмауэра.
- ❷ Создаем модуль базы данных, который использует паттерн «компоновщик» для построения необходимой группы серверов базы данных, сетей, балансировщиков нагрузки и межсетевых экранов.

- ③ Всегда создавайте группу серверов баз данных и сети, используя модуль фабрики. Модулю, использующему паттерн «строитель», нужна группа серверов баз данных.
- ④ Добавляем метод, позволяющий создать внутренний балансировщик нагрузки.
- ⑤ Добавляем метод, позволяющий создать внешний балансировщик нагрузки.
- ⑥ Добавляем метод, который позволяет создать правило брандмауэра, разрешающее трафик к базам данных.
- ⑦ Применяем модуль, использующий паттерн «компоновщик» для получения файла конфигурации JSON настроенных ресурсов базы данных.
- ⑧ Применяем модуль, использующий паттерн «строитель» базы данных для создания группы серверов базы данных с внешним доступом (балансировщик нагрузки и правило брандмауэра).
- ⑨ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

После запуска сценария Python вы получите длинную конфигурацию в формате JSON (листинг 3.9) с шаблонами экземпляров, группой серверов, менеджером группы серверов, внешним балансировщиком нагрузки и правилом брандмауэра. Модуль, использующий паттерн «строитель», генерирует все ресурсы, необходимые для создания базы данных, доступной извне. Имейте в виду, что для ясности в листинге опущены другие компоненты.

Листинг 3.9 Усеченная конфигурация системы базы данных

```
[
  {
    "google_compute_forwarding_rule": [
      {
        "db": [
          {
            "load_balancing_scheme": "EXTERNAL",
            "name": "development-database",
            "network_tier": "STANDARD",
            "port_range": "3306",
            "region": "us-central1",
            "target": "${google_compute_target_pool.db.id}"
          }
        ]
      }
    ],
  },
  {
    "google_compute_firewall": [
      {
        "db": [
          {
            "allow": [
              {
                "ports": [
                  "3306"
                ],
                "protocol": "tcp"
              }
            ]
          }
        ]
      }
    ]
  }
]
```



```

        }
      ],
      "name": "development-database",
      "network": "default",
    }
  ]
}
]

```

- ❶ Файл JSON определяет правило переадресации вычислений Google и брендмауэр с использованием ресурса Terraform. Для ясности в файле отсутствуют шаблоны экземпляров и группа серверов.
- ❷ Создаем балансировщик нагрузки со схемой EXTERNAL, что делает его доступным из внешнего источника.
- ❸ Создаем брендмауэр, который разрешает TCP-трафик через порт 3306 базы данных MySQL.
- ❹ Назначаем целью балансировщика нагрузки группу серверов базы данных. При этом используется встроенная функция интерполяции переменных Terraform для динамического разрешения идентификатора группы серверов базы данных.
- ❺ Файл JSON определяет правило переадресации вычислений Google и брендмауэр с использованием ресурса Terraform. Для ясности в файле не указаны шаблоны экземпляров и группа серверов.

Паттерн «строитель» помогает нам придерживаться принципа развиваемости. Вы можете выбрать набор необходимых вам ресурсов. Модуль с этим паттерном избавляет от необходимости настраивать правильную комбинацию атрибутов и ресурсов.

Кроме того, вы можете использовать модуль, применяющий паттерн «строитель», чтобы обернуть ресурсы облачного провайдера в обобщенный интерфейс. В примере кода Python есть методы построителя для `add_external_load_balancer`, которые охватывают правило переадресации вычислений GCP. Когда вы используете модуль, этот параметр описывает цель создания обобщенного балансировщика нагрузки, а не правила переадресации GCP.

Предметно-ориентированные языки

Некоторые DSL предлагают условный оператор `if-else` или цикл (итерацию), которые можно использовать в паттерне «строитель». Terraform предлагает аргумент `count` для создания заданного количества ресурсов на основе условного оператора. CloudFormation поддерживает условия для пользовательского ввода, которые могут выбирать стеки. Висер использует условия развертывания. В Ansible вы можете использовать условный импорт для выбора задач или сборников сценариев.

Например, вы можете настроить логическую переменную под названием `add_external_load_balancer`. Если вы передадите переменной значение `true`, DSL добавит условный оператор для создания внешнего

ресурса балансировщика нагрузки. В противном случае создается внутренний балансировщик нагрузки.

Некоторые DSL не предлагают условных операторов. Вам понадобится код, аналогичный примерам кода в этой книге, который служит паттерном DSL. Например, вы можете использовать Helm для создания паттернов и выпуска файлов YAML Kubernetes.

Паттерн «строитель» лучше всего применим к модулям, которые создают несколько ресурсов. К ним относятся модули, выполняющие настройку оркестраторов контейнеров, таких как Kubernetes, платформ с кластерной архитектурой, информационных панелей для метрик приложений и системы и т. д. Модуль, использующий паттерн «строитель», для этих случаев позволяет вам выбрать нужный ресурс без передачи определенных входных атрибутов.

Однако модули, использующие паттерн «строитель», могут быть сложными, поскольку они ссылаются на другие модули и несколько ресурсов. Риск неправильной конфигурации модуля может быть очень высоким. В главе 6 рассматриваются стратегии тестирования, позволяющие гарантировать функциональность и стабильность модуля.

3.6 Выбор паттерна

В этой главе я показала, как сгруппировать ресурсы системы баз данных в модулях, использующих различные паттерны. Но как выбрать, какой паттерн модуля использовать? А как насчет других ресурсов в системе баз данных, которые я не упомянула?

Вы можете создавать отдельные модули для новых ресурсов инфраструктуры с разными бизнес-функциями и целями. В примере с базой данных в этой главе конфигурация проекта Google (паттерн «одиночка»), сети (фабрика) и кластера базы данных (фабрика) разделена на модули. Каждый из них развивается как независимый ресурс с разными входными переменными и значениями по умолчанию.

В примере используется паттерн «компоновщик» для объединения всех модулей, использующих паттерны, в системе. Он использует фабричный паттерн для модулей сети, балансировщика нагрузки и кластера базы данных для передачи атрибутов и настройки каждого ресурса. Для тегов часто применяют паттерн «прототип», поскольку они содержат согласованные метаданные, копируемые в другие ресурсы. Для большинства написанных вами модулей вы будете использовать паттерны фабрики и прототипа, поскольку они обеспечивают возможность компоновки, воспроизводимости и развития.

Напротив, вы создаете проект Google, использующий паттерн «одиночка», потому что никто другой не будет изменять атрибуты отдельного экземпляра вашего проекта. Проект не сильно меняется,

поэтому вы используете паттерн попроще. Но вы решили сложную проблему создания системы баз данных с помощью паттерна «строитель». Модуль, использующий паттерн «строитель», позволяет вам выбирать конкретные ресурсы для создания.

На рис. 3.11 представлено дерево решений для определения того, какой паттерн использовать. Вы задаете себе ряд вопросов о назначении, частоте повторного использования и обновления, а также составе нескольких ресурсов. На основе этих критериев вы создаете модуль в соответствии с определенным паттерном.

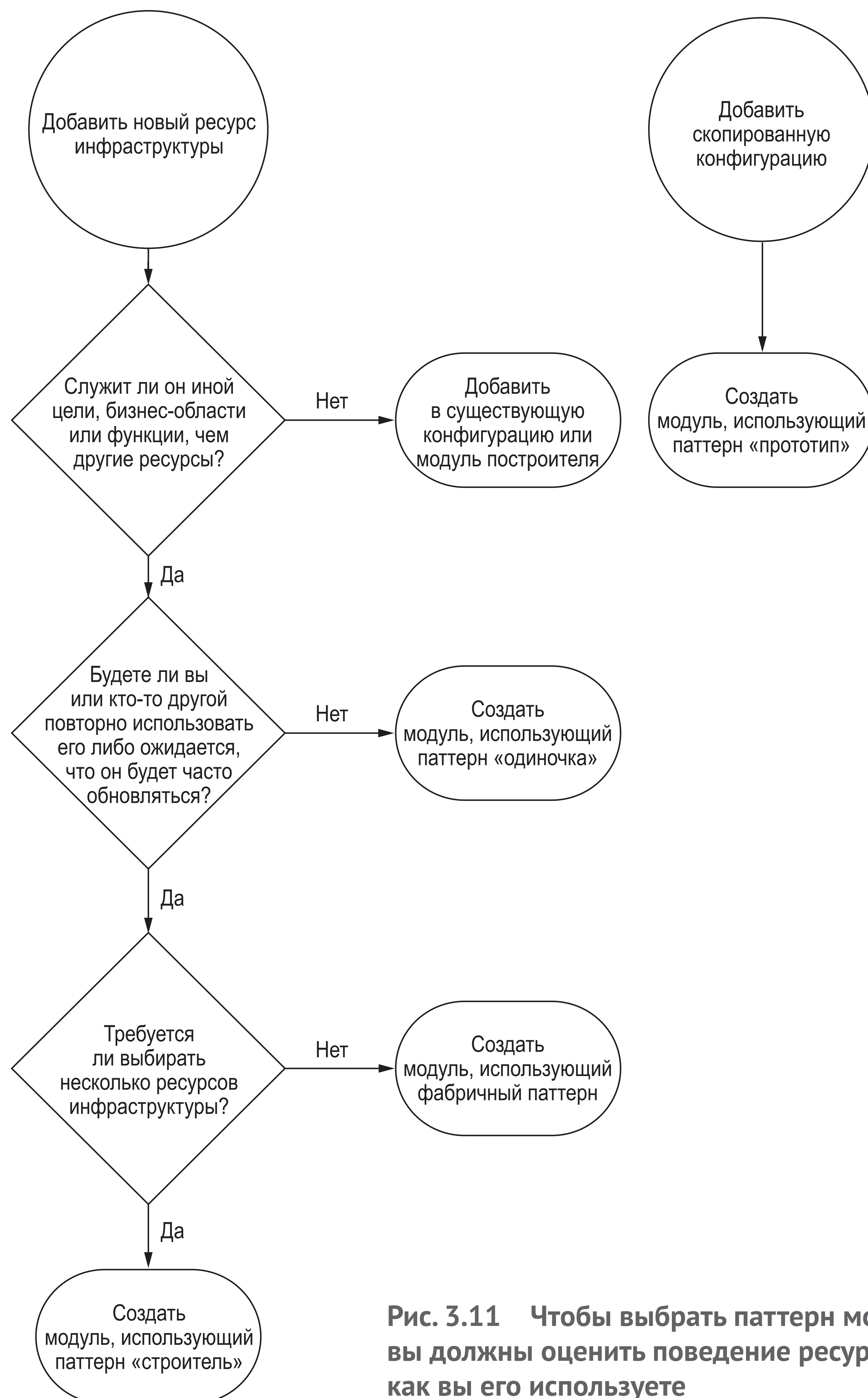


Рис. 3.11 Чтобы выбрать паттерн модуля, вы должны оценить поведение ресурса и то, как вы его используете

Следование дереву решений помогает создавать более komponуемые и развиваемые модули. Вам нужно сбалансировать предсказуемость стандартных атрибутов с гибкостью переопределения конфигурации для конкретных ресурсов. Однако будьте готовы к переменам. Модули могут перерасти ваши планы и изменить свои функции. Тот факт, что вы создаете модуль с одним паттерном, не означает, что вы не будете конвертировать его в другой в будущем!

Упражнение 3.1

К каким паттернам модулей применим следующий IaC (выберите все подходящие варианты):

```
if __name__ == "__main__" :  
    environment = 'development'  
    name = f'{environment}-hello-world'  
    cidr_block = '10.0.0.0/16'  
  
    # NetworkModule возвращает подсеть или сеть  
    network = NetworkModule(name, cidr_block)  
  
    # TagsModule возвращает список тегов по умолчанию  
    tags = TagsModule0  
  
    # ServerModule возвращает одиночный сервер  
    server = ServerModule(name, network, tags)
```

- a фабричный;
- b «одиночка»;
- c «прототип»;
- d «строитель»;
- e «компоновщик».

Ответы на упражнения приведены в приложении В.

Обратите внимание, что многие паттерны в этой главе ориентированы на создание модулей с помощью инструментов IaC. Иногда вы можете написать сценарий автоматизации на обычном языке программирования, потому что не удастся найти подходящие инструменты IaC. Такая ситуация чаще всего случается с устаревшей инфраструктурой. Например, представьте, что вам нужно создать систему баз данных в GCP. Однако у вас нет инструмента IaC, и вы можете получить доступ к API GCP только напрямую.

Чтобы создать систему баз данных с использованием API GCP, вы разделяете каждый ресурс инфраструктуры на фабричный модуль с четырьмя функциями: создание, чтение, обновление и удаление. Изменения ресурса опираются на эти функции. Вы можете проверить наличие ошибок в каждой функции в зависимости от действия, которое вы хотите выполнить для каждого ресурса.

На рис. 3.12 схематически показана реализация модулей фабрик для сервера, сети и балансировщика нагрузки. Каждый модуль содержит функционал создания, чтения, обновления и удаления. Модуль, отвечающий за работу с базой данных, использует паттерн «компоновщик» для создания, чтения, обновления и удаления одновременно всех ресурсов: сети, сервера и балансировщика нагрузки. Разделение изменений обновлений, которые вы вносите в ресурсы, на четыре функции упрощает автоматизацию. Даже паттерн «строитель» использует функции создания, чтения, обновления и удаления. Функции определяют автоматизированные действия, которые вы хотите использовать для настройки ресурсов. Но не забывайте проверять все функции на идемпотентность. Каждый запуск функции должен приводить вас к одной и той же конфигурации.

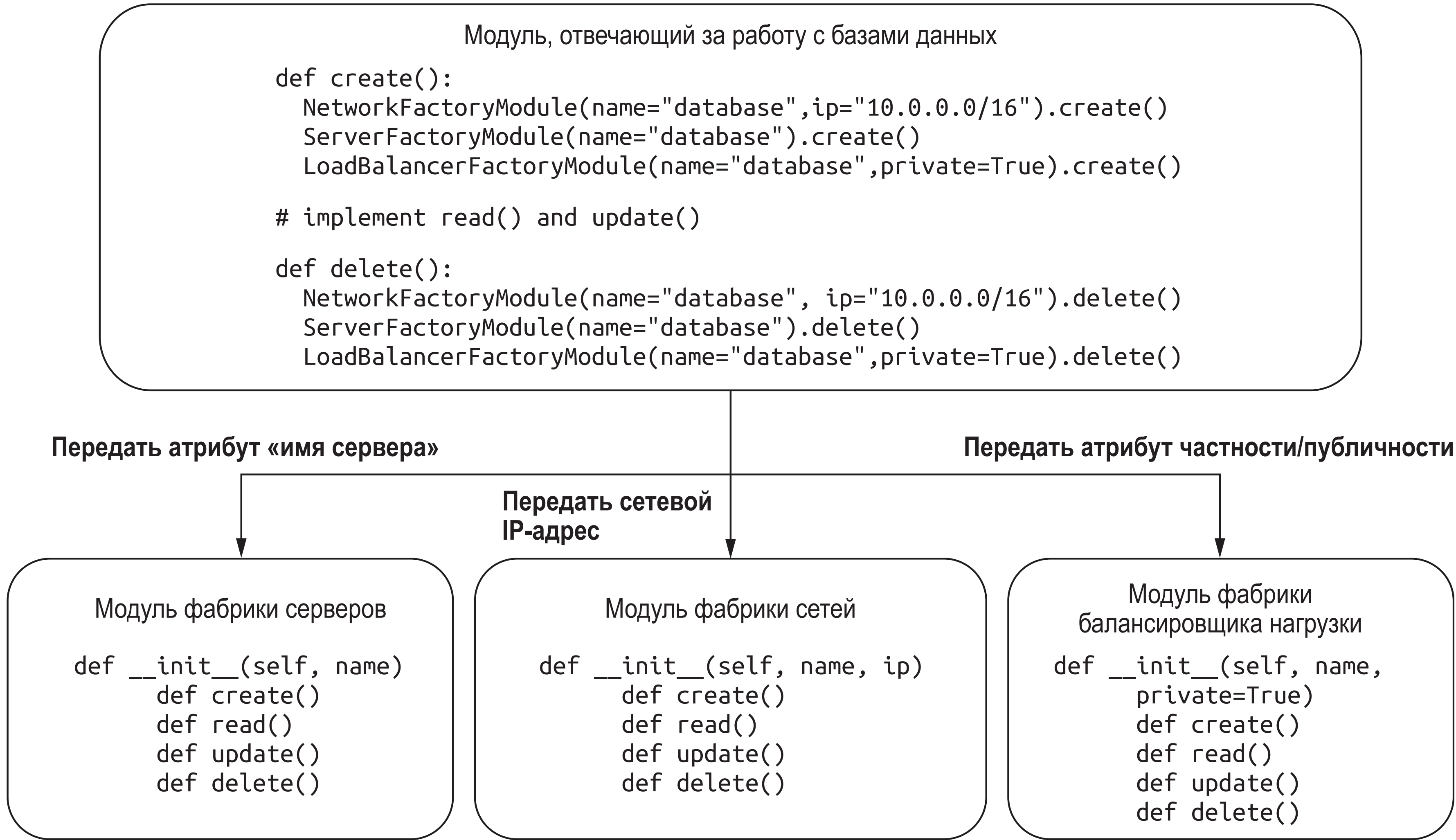


Рис. 3.12 Чтобы написать сценарии автоматизации, сначала создайте модули фабрик для отдельных ресурсов, а затем функции для создания, чтения, обновления и удаления ресурсов

Вы можете применить паттерны модулей, описанные в этой главе, для автоматизации и реализации IaC в любой инфраструктуре. Разрабатывая IaC, определите, где вы можете разбить свою инфраструктуру на модули. Принимая решение о том, когда и что следует делить на модули, учитывайте следующие аспекты:

- является ли ресурс общим?
- какие бизнес-задачи он решает?
- какая среда использует ресурс?

- какая команда управляет инфраструктурой?
- использует ли ресурс какой-либо особый инструмент?
- как изменить ресурс, не затрагивая что-либо еще в модуле?

Оценивая, какие ресурсы соответствуют различным бизнес-областям, командам или функциям, вы создаете меньшие наборы инфраструктуры. Как правило, модули стараются разрабатывать с минимально возможным количеством ресурсов. Модули с меньшим количеством ресурсов ускоряют подготовку и минимизируют ущерб от сбоя. Что еще более важно, вы можете развертывать, тестировать и отлаживать небольшие модули, прежде чем применять их в более обширной системе.

Группировка ресурсов в модули дает несколько преимуществ для вас, вашей команды и вашей компании. Для вас модули улучшают масштабируемость и отказоустойчивость ресурсов инфраструктуры. Вы минимизируете радиус воздействия изменений в модулях, чтобы повысить отказоустойчивость всей системы.

Для вашей команды модуль предоставляет преимущества механизма самообслуживания, позволяющего другим членам команды воспроизводить ваш модуль и создавать инфраструктуру. Ваш коллега по команде может использовать модуль и передавать переменные, которые он хочет настроить, вместо того чтобы искать и заменять признаки. Подробнее о совместном использовании модулей вы узнаете в главе 5.

С точки зрения вашей организации модуль поможет вам стандартизировать лучшие практики обеспечения безопасности и построения инфраструктуры для всех ресурсов. Вы можете использовать одни и те же конфигурации, чтобы не создавать вручную одинаковые балансировщики нагрузки и ограничения в правилах брандмауэра. Модули также помогают вашей службе безопасности проводить аудит и применять эти практики в разных командах, как описано в главе 8.

Краткое содержание главы

- Применяйте различные паттерны модулей, такие как «одиночка», фабричный, «прототип» и «строитель», чтобы можно было создавать компокуемые конфигурации инфраструктуры.
- Используйте паттерн «компоновщик» для группировки ресурсов инфраструктуры в иерархию и структурирования их для автоматизации.
- Для управления отдельными экземплярами инфраструктуры можно использовать паттерн «одиночка», который применяется к редко изменяющимся ресурсам инфраструктуры.
- Используйте паттерн «прототип» для копирования и применения глобальных параметров конфигурации, таких как теги или общая конфигурация.

- Модули фабрики принимают входные данные для создания ресурсов инфраструктуры с определенной конфигурацией.
- Модули, использующие паттерн «строитель», принимают входные данные, чтобы решить, какие ресурсы создавать. Такие модули могут состоять из модулей фабрик.
- При принятии решения о том, что и как следует разбить на модули, оцените, какие функции или бизнес-задачи обслуживает конфигурация инфраструктуры.
- Если вы пишете сценарии для автоматизации инфраструктуры, создайте модули фабрик с возможностями создания, чтения, обновления и удаления. Ссылайтесь на них в модулях, использующих паттерн «строитель».

Паттерны инфраструктурных зависимостей

Основные темы главы:

- разработка слабосвязанных модулей инфраструктуры с использованием паттернов зависимостей;
- определение способов разделения инфраструктурных зависимостей;
- выбор вариантов использования инфраструктуры для паттернов зависимостей.

Инфраструктурная система состоит из набора ресурсов, которые зависят друг от друга. Например, сервер зависит от существования сети. Как узнать, что сеть существует, прежде чем создавать сервер? Вы можете выполнить проверку с помощью инфраструктурной зависимости, которая возникает, когда для создания или изменения одного ресурса требуется наличие другого ресурса.

ОПРЕДЕЛЕНИЕ *Инфраструктурная зависимость* выражает отношения, в которых ресурс инфраструктуры зависит от существования и атрибутов другого ресурса.

Обычно зависимость сервера от сети определяется путем жесткого кодирования идентификатора сети. Однако жесткое кодирование

образует слишком сильную зависимость сервера от сети. Каждый раз, когда вы меняете сеть, вы должны обновить жестко запрограммированную зависимость.

В главе 2 вы узнали, как избегать жесткого кодирования значений с помощью переменных, чтобы лучше придерживаться принципов воспроизводимости и развиваемости. Передача идентификатора сети в качестве переменной позволяет лучше разделить сервер и сеть. Однако переменная связывает ресурсы только в одном модуле. Как выразить зависимости между модулями?

Предыдущая глава сгруппировала ресурсы в модули для повышения композиционности. В этой главе рассматриваются паттерны управления зависимостями инфраструктуры для повышения ее адаптивности (изменчивости). Заменить один модуль другим проще при наличии слабой зависимости.

На самом деле инфраструктурные системы бывают довольно сложными, и вы не сможете заменять любые модули совсем без сбоев. Слабосвязанные зависимости позволяют снизить риск сбоя при обновлении, но не гарантируют 100%-ную доступность!

4.1 Однонаправленные отношения

На изменение инфраструктуры могут влиять зависимости разного типа. Представьте, что вы добавляете правило брандмауэра каждый раз, когда создаете новое приложение. Правило брандмауэра имеет однонаправленную зависимость от IP-адреса приложения, разрешая трафик. Любое изменение приложения отражается в правиле брандмауэра.

ОПРЕДЕЛЕНИЕ *Однонаправленная зависимость* (unidirectional dependency) выражает одностороннюю связь, при которой только один ресурс ссылается на другой.

Вы можете выражать однонаправленные зависимости между любым набором ресурсов или модулей. Рисунок 4.1 описывает однонаправленную зависимость между правилом брандмауэра и приложением. Правило зависит от приложения, поэтому оно находится выше в стеке инфраструктуры, чем приложение более низкого уровня.

Когда вы выражаете зависимость, у вас есть ресурс верхнего уровня, такой как брандмауэр, который зависит от существования ресурса нижнего уровня, такого как приложение.

ОПРЕДЕЛЕНИЕ *Ресурс верхнего уровня* зависит от другого ресурса или модуля. *Ресурс нижнего уровня* имеет зависящие от него ресурсы высокого уровня.

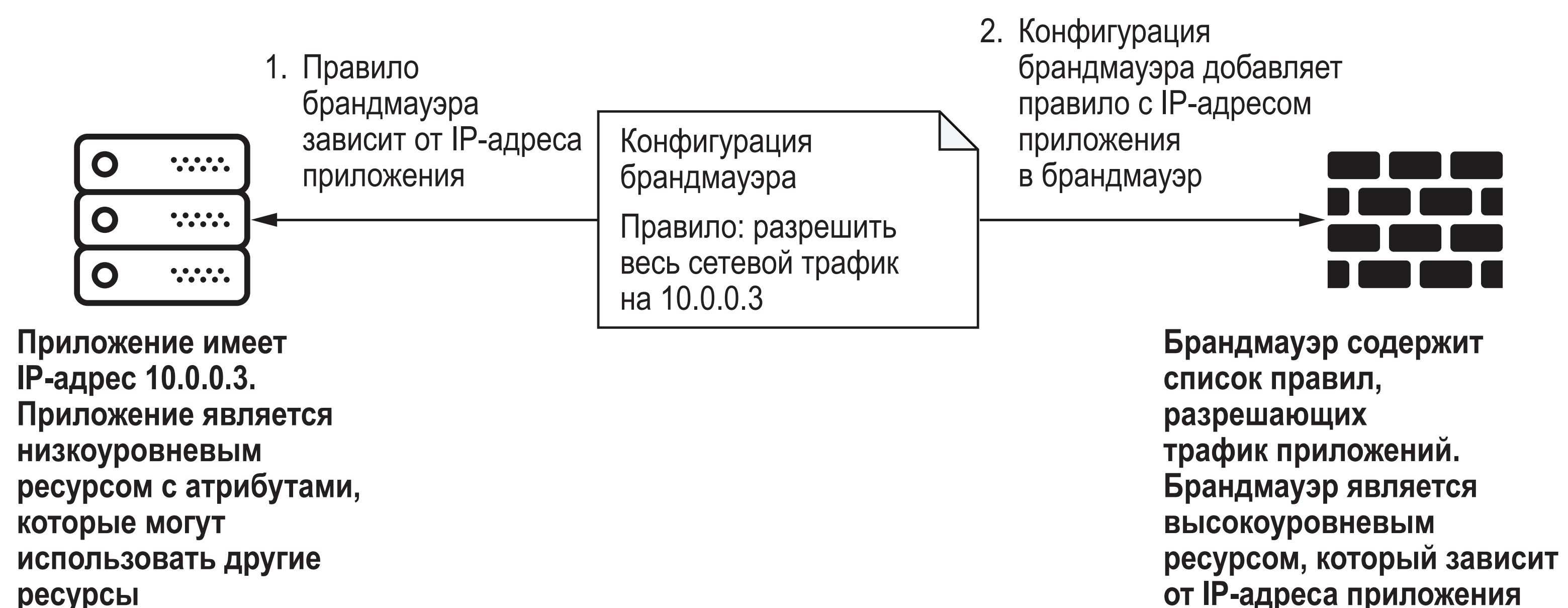


Рис. 4.1 Правило брандмауэра однонаправленно зависит от IP-адреса приложения

Допустим, приложению для создания отчетов нужен список правил брандмауэра. Оно отправляет правила в приложение аудита. Однако брандмауэру необходимо знать IP-адрес приложения, отправляющего отчеты. Что следует обновить в первую очередь – правила брандмауэра или IP-адрес приложения для создания отчетов? Рисунок 4.2 иллюстрирует проблему выбора, какое приложение следует обновить в первую очередь.

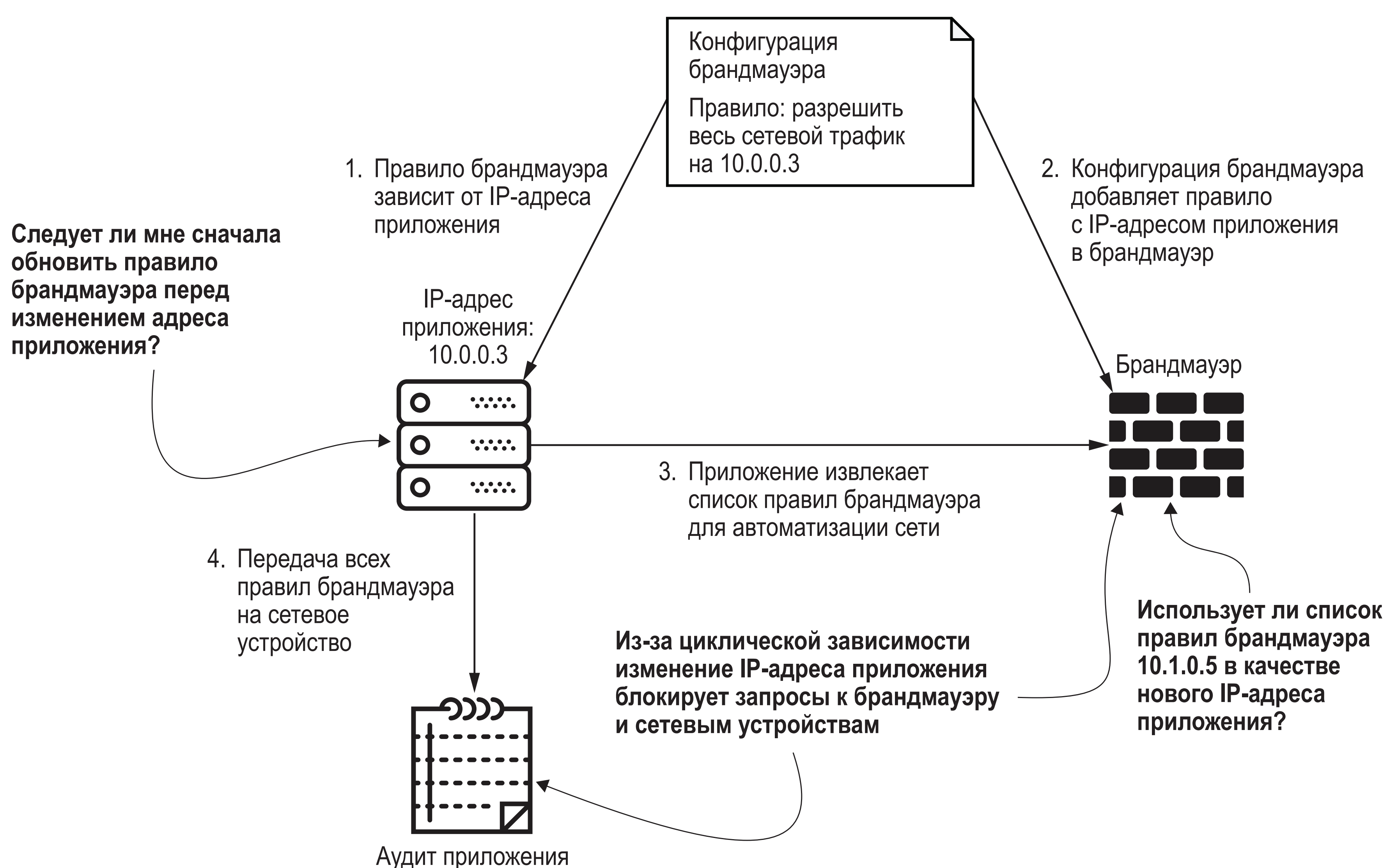


Рис. 4.2 Приложение для создания отчетов и брандмауэр имеют циклическую зависимость друг от друга. Изменения блокируют подключение к приложению

В этом примере встречается циклическая зависимость – своего рода проблема курицы и яйца. Вы не можете изменить один ресурс,

не затрагивая другой. Если вы сначала измените адрес приложения для создания отчетов, правило брандмауэра должно измениться. Однако приложение для создания отчетов выходит из строя, поскольку оно не может подключиться. Возможно, вы заблокировали его запрос!

Циклические зависимости вызывают неожиданное поведение во время изменений, что в конечном итоге влияет на возможность компоновки и развития. Вы не знаете, какой ресурс обновить в первую очередь. Напротив, вы можете определить, как изменение модуля низкого уровня может повлиять на модуль высокого уровня. Отношения однонаправленной зависимости делают изменения более предсказуемыми. В конце концов, успешное изменение инфраструктуры зависит от двух факторов: предсказуемости и изоляции.

4.2 Внедрение зависимостей

Однонаправленные зависимости помогают минимизировать влияние изменений модуля низкого уровня на модули высокого уровня. Например, изменения в сети не должны нарушать работу ресурсов высокого уровня, таких как очереди, приложения или базы данных. В этом разделе применяется концепция внедрения зависимостей при разработке программного обеспечения в инфраструктуру и последующего разделения однонаправленных зависимостей. Внедрение зависимостей основано на двух принципах: инверсии управления и инверсии зависимостей.

4.2.1 Инверсия управления

При наличии однонаправленных связей в зависимостях инфраструктуры ваш ресурс высокого уровня получает информацию о ресурсе низкого уровня. Затем он может запустить свои изменения. Например, сервер получает информацию об идентификаторе сети и диапазоне IP-адресов, прежде чем запросить IP-адрес (рис. 4.3).

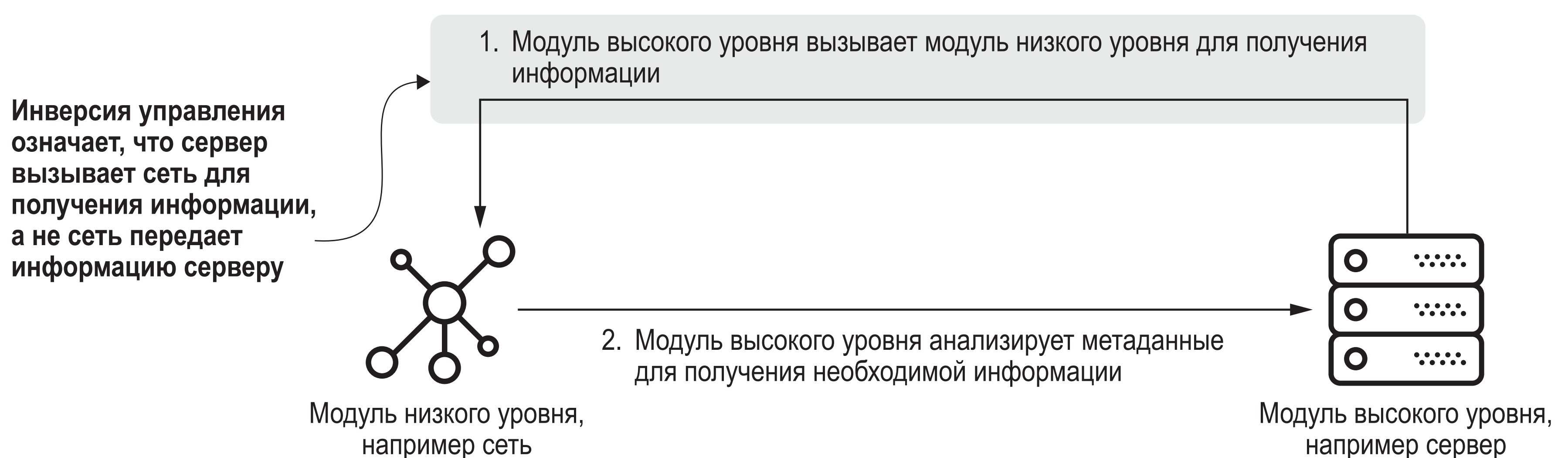


Рис. 4.3 При инверсии управления ресурс или модуль высокого уровня обращается к модулю низкого уровня для получения информации и анализирует его метаданные на предмет любых зависимостей

Сервер обращается к сети, применяя принцип разработки программного обеспечения, называемый инверсией управления. Ресурс высокого уровня запрашивает информацию о ресурсе низкого уровня перед обновлением.

ОПРЕДЕЛЕНИЕ *Инверсия управления* (inversion of control) – это принцип, по которому ресурс высокого уровня обращается к ресурсу низкого уровня за атрибутами или ссылками.

Вот простой пример из жизни: вы используете инверсию управления, когда звоните, чтобы записаться на прием к врачу, вместо того чтобы врач автоматически назначал вам прием.

Теперь применим инверсию управления для реализации зависимости сервера от сети. Вы создаете сеть с помощью сетевого модуля. В листинге 4.1 сетевой модуль выводит имя сети и сохраняет его в файле с именем terraform.tfstate. Ресурсы высокого уровня, такие как сервер, могут анализировать имя сети из этого файла JSON.

Листинг 4.1 Выходные данные сетевого модуля в файле JSON

```
{
  "outputs": {
    "name": {
      "value": "hello-world-subnet",
      "type": "string"
    }
  }
}
```

- ❶ При создании сети с помощью Terraform генерируется файл JSON со списком выходных данных. Terraform использует этот файл для отслеживания создаваемых им ресурсов.
- ❷ Сетевой модуль выводит имя подсети в виде строки.
- ❸ Оставшаяся часть файла JSON опущена для ясности.

Следуя принципу инверсии управления, сервер открывает сетевой файл terraform.tfstate (листинг 4.2) и считывает имя подсети. Поскольку модуль предоставляет выходные данные в файле JSON, вашему серверному модулю необходимо прочитать его и извлечь значение имени подсети (hello-world-subnet).

Листинг 4.2 Применение инверсии управления для создания сервера в сети

```
import json

class NetworkModuleOutput:
    def __init__(self):
        with open('network/terraform.tfstate', 'r') as network_state:
            network_attributes = json.load(network_state)
```



```

        self.name = network_attributes['outputs']['name']['value'] ❷

class ServerFactoryModule: ❸
    def __init__(self, name, zone='us-central1-a'): ❹
        self._name = name
        self._network = NetworkModuleOutput() ❺
        self._zone = zone
        self.resources = self._build() ❻

    def _build(self): ❻
        return {
            'resource': [{
                'google_compute_instance': [{ ❹
                    self._name: [{
                        'allow_stopping_for_update': True,
                        'boot_disk': [{
                            'initialize_params': [{
                                'image': 'ubuntu-1804-lts'
                            }]
                        }],
                    },
                    'machine_type': 'e2-micro', ❹
                    'name': self._name, ❹
                    'zone': self._zone,
                    'network_interface': [{
                        'subnetwork': self._network.name ❷
                    }]
                }]
            }]
        }

if __name__ == "__main__": ❸
    server = ServerFactoryModule(name='hello-world') ❸
    with open('main.tf.json', 'w') as outfile: ❸
        json.dump(server.resources, outfile, sort_keys=True, indent=4)

```

- ❶ Создаем объект, который отражает схему вывода сетевого модуля. Это облегчает серверу получение имени подсети.
- ❷ Объект сетевого вывода анализирует значение имени подсети из объекта JSON.
- ❸ Создаем модуль сервера, использующий фабричный паттерн.
- ❹ Создаем экземпляр машины Google, используя ресурс Terraform с именем и зоной.
- ❺ Модуль сервера вызывает объект сетевого вывода, который содержит имя подсети, полученное из файла JSON сетевого модуля.
- ❻ Применяем модуль для создания конфигурации JSON сервера с использованием имени подсети.
- ❼ Модуль сервера ссылается на имя сетевого вывода и передает его в поле subnetwork.
- ❽ Сохраняем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

В AWS вы должны использовать ресурс Terraform `aws_instance` со ссылкой на сеть, которую вы хотите применить (<http://mng.bz/PnPR>). В Azure используйте ресурс Terraform `azurerm_linux_virtual_machine` (<http://mng.bz/J2DZ>) в сети.

Реализация инверсии управления исключает прямую ссылку на подсеть в вашем серверном модуле. У вас есть возможность контролировать и ограничивать информацию, возвращаемую сетью для использования ресурсами высокого уровня. Что еще более важно, вы расширите возможности компоновки, поскольку сможете создавать другие серверы и ресурсы высокого уровня в подсети с именем, предлагаемым сетевым модулем.

Что, если другим ресурсам высокого уровня потребуются другие атрибуты низкого уровня? Например, вы можете создать очередь, которой нужен диапазон IP-адресов подсети. Чтобы решить эту проблему, вы развиваете сетевой модуль, добавляя в него вывод диапазона IP-адресов подсети. Очередь может извлечь нужный ей диапазон из выходных данных модуля.

Инверсия управления улучшает развиваемость, поскольку ресурсы высокого уровня нуждаются в разных атрибутах. Вы можете развивать ресурсы низкого уровня, не переписывая IaC для ресурсов высокого уровня. Однако вам нужен способ защитить ресурсы высокого уровня от любых обновлений атрибутов или переименования ресурсов низкого уровня.

4.2.2 Инверсия зависимостей

Хотя инверсия управления обеспечивает развиваемость модулей высокого уровня, она не защищает их от изменений в модулях низкого уровня. Предположим, вы меняете имя сети на ее идентификатор. Когда вы внесете очередные обновления в серверный модуль, он выйдет из строя! Серверный модуль не распознает идентификатор сети.

Чтобы защитить серверный модуль от изменений информации о сети, мы добавляем уровень абстракции между источником информации о сети и сервером. На рис. 4.4 сервер получает доступ к атрибутам сети через API или сохраненную конфигурацию, вместо того чтобы извлекать их из сетевых выходных данных. Все эти интерфейсы служат абстракциями для получения сетевых метаданных.

С помощью инверсии зависимостей вы можете изолировать изменения в модулях низкого уровня и минимизировать нарушение

их зависимостей. Инверсия зависимостей требует, чтобы ресурсы высокого и низкого уровней имели зависимости, выраженные через абстракции.



Рис. 4.4 Инверсия зависимостей возвращает абстракцию метаданных ресурса низкого уровня ресурсу верхнего уровня, который от него зависит

ОПРЕДЕЛЕНИЕ *Инверсия зависимостей* (dependency inversion) – это принцип выражения зависимостей между модулями или ресурсами высокого и низкого уровней через абстракции.

Уровень абстракции действует как переводчик или посредник, который передает необходимые атрибуты. Он служит буфером для изменений в модуле низкого уровня относительно модуля высокого уровня. В общем случае вы можете выбрать один из трех типов абстракции:

- интерполяция атрибутов ресурсов (внутри модулей);
- выходные данные модулей (между модулями);
- состояние инфраструктуры (между модулями).

Некоторые абстракции, такие как интерполяция атрибутов или выходные данные модуля, зависят от вашего инструмента. Абстракция по состоянию инфраструктуры будет зависеть от вашего инструмента или API инфраструктуры. На рис. 4.5 показаны абстракции с помощью интерполяции атрибутов, выходных данных модуля или состояния инфраструктуры для передачи сетевых метаданных на сервер.

В листинге 4.3 показан пример реализации трех типов абстракции путем создания модулей для сети и сервера. Я начну с интерполяции атрибутов. Интерполяция атрибутов обеспечивает передачу атрибутов между ресурсами или задачами внутри модуля или конфигурации. Используя Python, подсеть интерполирует имя сети, обращаясь к атрибуту имени, присвоенному сетевому объекту.



Рис. 4.5 В зависимости от инструмента и зависимостей абстракции для инверсии зависимостей могут использовать интерполяцию атрибутов, выходные данные модуля или состояние инфраструктуры

Листинг 4.3 Использование интерполяции атрибутов для получения имени сети

```
import json

class Network:
    def __init__(self, name="hello-network"):
        self.name = name
        self.resource = self._build()

    def _build(self):
        return {
            'google_compute_network': [
                {
                    f'{self.name}': [
                        {
                            'name': self.name
                        }
                    ]
                }
            ]
        }

class Subnet:
    def __init__(self, network, region='us-central1'):
        self.network = network
        self.name = region
        self.subnet_cidr = '10.0.0.0/28'
        self.region = region
        self.resource = self._build()

    def _build(self):
```



```

    return {
        'google_compute_subnetwork': [
            {
                f'{self.name}': [
                    {
                        'name': self.name,
                        'ip_cidr_range': self.subnet_cidr,
                        'region': self.region,
                        'network': self.network.name
                    }
                ]
            }
        ]
    }

if __name__ == "__main__" :
    network = Network()
    subnet = Subnet(network)

    resources = {
        "resource": [
            network.resource
            subnet.resource
        ]
    }

    with open(f'main.tf.json', 'w') as outfile:
        json.dump(resources, outfile, sort_keys=True, indent=4)

```

- ❶ Создаем сеть Google, используя ресурс Terraform, с именем hello-network.
- ❷ Применяем модуль для создания конфигурации JSON сети.
- ❸ Создаем подсеть Google, используя ресурс Terraform, названный по имени региона us-central1.
- ❹ Передаем весь сетевой объект в подсеть. Подсеть вызывает сетевой объект для получения необходимых ей атрибутов.
- ❺ Интерполируем имя сети, получая его из объекта.
- ❻ Применяем модуль для создания конфигурации JSON сети.
- ❼ Применяем модуль для создания конфигурации JSON подсети и передаем сетевой объект в подсеть.
- ❽ Объединяем объекты JSON сети и подсети в структуру JSON, совместимую с Terraform.
- ❾ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Предметно-ориентированные языки

Инструменты IaC, использующие DSL, предлагают собственный формат интерполяции переменных. В примере Terraform для динамической передачи имени сети в подсеть будет использоваться `google_compute_network.hello-world-network.name`. CloudFormation позволяет ссылаться на параметры с помощью `Ref`. В Vicer вы можете ссылаться на свойства ресурса `properties`.

Интерполяция атрибутов работает между модулями или ресурсами в конфигурации. Однако интерполяция работает только для определенных инструментов и не обязательно для всех ресурсов. Когда у вас скомпоновано много ресурсов и модулей, вы не можете использовать интерполяцию.

Одна из альтернатив интерполяции атрибутов использует явные выходные данные модуля для передачи атрибутов ресурсов между модулями. Вы можете настроить выходные данные для любой схемы или параметров, которые вам нужны. Например, вы можете сгруппировать подсеть и сеть в один модуль и экспортировать их атрибуты для использования сервером. Давайте проведем рефакторинг подсети и сети и добавим сервер, как показано в листинге 4.4.

Листинг 4.4 Установка имени подсети в качестве выходных данных для модуля

```
import json 1
class NetworkModule: 2
    def __init__(self, region='us-central1'):
        self._region = region
        self._network = Network() 2
        self._subnet = Subnet(self._network) 2
        self.resource = self._build() 3
    def _build(self): 3
        return [ 3
            self._network.resource, 3
            self._subnet.resource 3
        ] 3
class Output: 4
    def __init__(self, subnet_name): 4
        self.subnet_name = subnet_name 4
    def output(self): 5
        return self.Output(self._subnet.name) 5
class ServerModule: 6
    def __init__(self, name, network, 7
        zone='us-central1-a'):
        self._name = name
        self._subnet_name = network.subnet_name 8
        self._zone = zone
        self.resource = self._build() 9
    def _build(self): 9
        return [{
            'google_compute_instance': [{
```



```

        self._name: [{
            'allow_stopping_for_update': True,
            'boot_disk': [{
                'initialize_params': [{
                    'image': 'ubuntu-1804-lts'
                }]
            }],
            'machine_type': 'e2-micro',
            'name': self._name,
            'zone': self._zone,
            'network_interface': [{
                'subnetwork': self._subnet_name
            }]
        }]
    }
}

if __name__ == "__main__":
    network = NetworkModule() 10
    server = ServerModule("hello-world", 6
                           network.output()) 7
    resources = { 11
        "resource": network.resource + server.resource 11
    } 11

    with open(f'main.tf.json', 'w') as outfile: 12
        json.dump(resources, outfile, sort_keys=True, indent=4) 12

```

- ❶ Объекты сети и подсети опущены для ясности.
- ❷ Переносим создание сети и подсети в модуль. Этот подход соответствует паттерну «компоновщик». Модуль создает сеть и подсеть Google, используя ресурсы Terraform.
- ❸ Применяем модуль для создания конфигурации JSON сети и подсети.
- ❹ Создаем вложенный класс для вывода сетевого модуля. Вложенный класс экспортирует имя подсети для использования атрибутами высокого уровня.
- ❺ Создаем функцию вывода сетевого модуля для получения и экспорта всех сетевых выходных данных.
- ❻ Этот модуль создает вычислительный экземпляр (сервер) Google с использованием ресурса Terraform.
- ❼ Передаем сетевые выходные данные в качестве входной переменной для серверного модуля. Сервер выберет необходимые ему атрибуты.
- ❽ С помощью объекта вывода данных о сети получаем имя подсети и присваиваем ему атрибут имени подсети сервера.
- ❾ Применяем модуль для создания конфигурации JSON сервера.
- ❿ Переносим создание сети и подсети в модуль. Этот подход соответствует паттерну «компоновщик». Модуль создает сеть и подсеть Google, используя ресурсы Terraform.
- ⓫ Объединяем объекты JSON сети и сервера в структуру JSON, совместимую с Terraform.
- ⓫ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Предметно-ориентированные языки

Если вы работаете с такими инструментами подготовки, как CloudFormation, Vicer или Terraform, то генерируете выходные данные для модулей или стеков, которые могут использовать модули более высокого уровня. Инструмент управления конфигурацией, такой как Ansible, передает переменные посредством стандартного вывода между задачами автоматизации.

Выходные данные модуля содержат нужные параметры для ресурсов высокого уровня. В этом подходе мы копируем и повторяем значения. Однако выходные данные модуля могут оказаться запутанными! На практике вы будете часто забывать, какие результаты вы предоставили и их названия. Контрактное тестирование, описанное в главе 6, направлено на соблюдение требований к выходным данным модуля.

Вместо применения выходных данных вы можете использовать состояние инфраструктуры в виде файла состояния или метаданных API провайдера инфраструктуры. Многие инструменты хранят копию состояния инфраструктуры, которую я называю *инструментальным состоянием*, чтобы обнаруживать расхождения между фактическим состоянием ресурса и конфигурацией и отслеживать ресурсы, на которые влияет инструмент.

ОПРЕДЕЛЕНИЕ *Инструментальное состояние (tool state)* – это представление состояния инфраструктуры, хранимое инструментом IaC. Оно отслеживает конфигурацию ресурсов, управляемых инструментом.

Инструменты часто сохраняют свое состояние в файле. Вы уже встречали пример использования инструментального состояния в листинге 4.2. Вы проанализировали имя сети из файла terraform.tfstate, который является состоянием инструмента для Terraform. Однако не все инструменты сохраняют файл состояния. В результате у вас могут возникнуть трудности с анализом атрибутов ресурсов низкого уровня в разных инструментах.

Если в вашей системе имеется несколько инструментов и поставщиков, у вас есть два основных варианта решения этой проблемы. Прежде всего можно использовать в качестве стандартного интерфейса для передачи метаданных так называемый *менеджер конфигурирования* (configuration manager, система управления конфигурацией). Эта система, как и хранилище значений ключей, управляет набором полей и их значений.

Менеджер конфигурирования помогает нам создать собственный уровень абстракции для инструментального состояния. Например, некоторые сценарии сетевой автоматизации могут считывать зна-

чения IP-адреса, хранящиеся в хранилище значений ключа. Однако вам необходимо иметь менеджер конфигурирования и убедиться, что ваш IaC имеет к нему доступ.

В качестве второго варианта попробуйте использовать API поставщика инфраструктуры. API инфраструктуры меняются нечасто; они предоставляют подробную информацию и учитывают изменения, которые не всегда включаются в файл состояния. Для доступа к информации из API инфраструктуры можно применить клиентские библиотеки.

Предметно-ориентированные языки

Многие инструменты конфигурирования позволяют выполнять вызовы API инфраструктуры. Например, типы параметров, специфичные для AWS, и `Fn::ImportValue` в CloudFormation извлекают значения из API AWS или других стеков. Bicep предлагает ключевое слово `existing` для импорта свойств ресурса за пределы текущего файла.

Terraform предлагает источники данных для чтения метаданных ресурса инфраструктуры из API. Аналогично модуль может ссылаться на факты¹ Ansible, которые собирают метаданные о ресурсе или вашей среде.

Вы неизбежно столкнетесь с некоторыми недостатками использования API инфраструктуры. К сожалению, вашему IaC требуется доступ к сети. Вы не узнаете *значение* атрибута, пока не запустите IaC, поскольку код должен отправить запрос к API. Если в API инфраструктуры произойдет сбой, ваш IaC может не получить атрибуты для ресурсов низкого уровня.

Когда вы добавляете абстракцию с инверсией зависимостей, вы защищаете ресурсы высокого уровня от изменения атрибутов ресурсов более низкого уровня. Хотя вы не можете предотвратить все сбои, вы минимизируете потенциальный ущерб благодаря обновлению ресурсов низкого уровня. Это можно рассматривать как своего рода договоренность: если ресурсы высокого и низкого уровней согласовывают необходимые им атрибуты, они могут развиваться независимо друг от друга.

4.2.3 Применение внедрения зависимостей

Что произойдет, если объединить инверсию управления и инверсию зависимостей? На рис. 4.6 показано, как можно применить оба принципа, чтобы разделить сервер и сеть в нашем примере. Сервер обращается к сети для получения атрибутов и анализирует метаданные, используя API инфраструктуры или состояние. Если вы

¹ В Ansible *факты* – это переменные, содержащие данные об управляемых системах. – Прим. перев.

вносите изменения в имя сети, метаданные обновляются. Сервер получает обновленные метаданные и отдельно настраивает их конфигурацию.

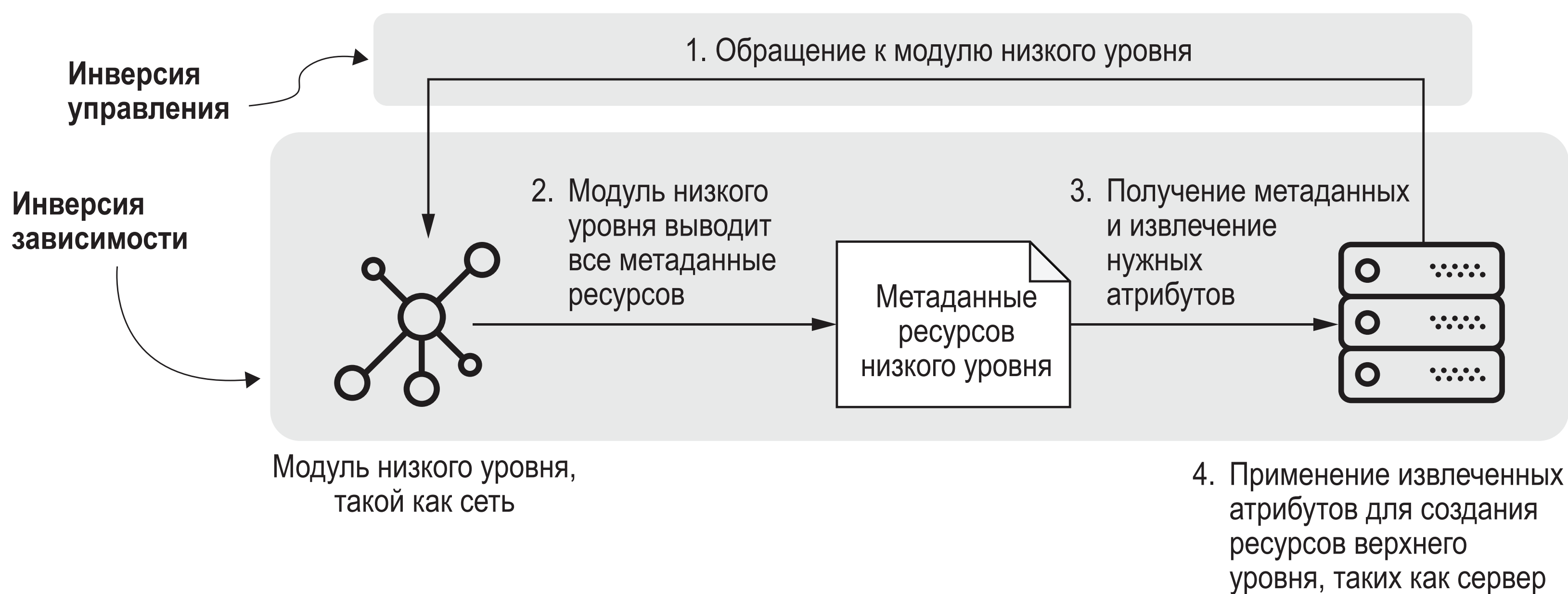


Рис. 4.6 Внедрение зависимостей применяется для ослабления инфраструктурных зависимостей и изоляции ресурсов низкого и высокого уровней

Использование обоих принципов способствует повышению развиваемости и компонуемости, поскольку уровень абстракции действует как буфер между каждой составной частью вашей системы. Внедрение зависимостей объединяет инверсию управления и инверсию зависимостей. Инверсия управления изолирует изменения в модулях или ресурсах высокого уровня, тогда как инверсия зависимостей изолирует изменения в ресурсах низкого уровня.

ОПРЕДЕЛЕНИЕ Внедрение зависимостей (dependency injection) сочетает в себе принципы инверсии управления и инверсии зависимостей. Модули или ресурсы высокого уровня запрашивают атрибуты из модулей низкого уровня через абстракцию.

Теперь реализуем внедрение зависимостей для примера сервера и сети с помощью Apache Libcloud – библиотеки для GCP API, – как показано в листинге 4.5. Для поиска сети мы применим Libcloud. Сервер вызывает API GCP для получения имени подсети, анализирует метаданные API GCP и назначает себе пятый IP-адрес в диапазоне сети.

Листинг 4.5 Использование внедрения зависимостей для создания сервера в сети

```
import credentials
import ipaddress
import json
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
```

1

1


```

def get_network(name):
    ComputeEngine = get_driver(Provider.GCE)
    driver = ComputeEngine(
        credentials.GOOGLE_SERVICE_ACCOUNT,
        credentials.GOOGLE_SERVICE_ACCOUNT_FILE,
        project=credentials.GOOGLE_PROJECT,
        datacenter=credentials.GOOGLE_REGION)
    return driver.ex_get_subnetwork(
        name, credentials.GOOGLE_REGION)

class ServerFactoryModule:
    def __init__(self, name, network, zone='us-central1-a'):
        self._name = name
        gcp_network_object = get_network(network)
        self._network = gcp_network_object.name
        self._network_ip = self._allocate_fifth_ip_address_in_range(
            gcp_network_object.cidr)
        self._zone = zone
        self.resources = self._build()

    def _allocate_fifth_ip_address_in_range(self, ip_range):
        ip = ipaddress.IPv4Network(ip_range)
        return format(ip[5])

    def _build(self):
        return {
            'resource': [{
                'google_compute_instance': [{
                    self._name: [{
                        'allow_stopping_for_update': True,
                        'boot_disk': [{
                            'initialize_params': [{
                                'image': 'ubuntu-1804-lts'
                            }]
                        }],
                    }],
                'machine_type': 'e2-micro',
                'name': self._name,
                'zone': self._zone,
                'network_interface': [{
                    'subnetwork': self._network,
                    'network_ip': self._network_ip
                }]
            }]}
        ]}

if __name__ == "__main__":
    server = ServerFactoryModule(name='hello-world', network='default')
    with open('main.tf.json', 'w') as outfile:
        json.dump(server.resources, outfile, sort_keys=True, indent=4)

```


- ❶ Импортируем библиотеку Libcloud, которая позволяет получить доступ к API GCP. Вам необходимо импортировать объект провайдера и драйвер Google.
- ❷ Эта функция получает информацию о сети с помощью библиотеки Libcloud. Сеть и подсеть создавались отдельно. Их код опущен для ясности.
- ❸ Импортируем драйвер Google Compute Engine для Libcloud.
- ❹ Передаем учетные данные службы GCP, которые Libcloud будет использовать для доступа к API GCP.
- ❺ Используем драйвер Libcloud для получения информации о подсети по ее имени.
- ❻ Этот модуль создает вычислительный экземпляр (сервер) Google с использованием ресурса Terraform.
- ❼ Извлекаем имя подсети из сетевого объекта GCP, возвращенного Libcloud, и используем его для создания сервера.
- ❽ Извлекаем блок CIDR из сетевого объекта GCP, возвращенного Libcloud, и используем его для вычисления пятого IP-адреса в сети. Сервер использует результат в качестве своего сетевого IP-адреса.
- ❾ Применяем модуль для создания конфигурации JSON сервера.
- ❿ Извлекаем имя подсети из сетевого объекта GCP, возвращенного Libcloud, и используем его для создания сервера.
- ⓫ Извлекаем блок CIDR из сетевого объекта GCP, возвращенного Libcloud, и используем его для вычисления пятого IP-адреса в сети. Сервер использует результат в качестве своего сетевого IP-адреса.
- ⓬ Этот модуль создает вычислительный экземпляр (сервер) Google с использованием ресурса Terraform.
- ⓭ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Чтобы преобразовать листинг 4.5, необходимо обновить IaC и создать экземпляр Amazon Elastic Compute Cloud (EC2) или виртуальную машину Azure Linux. Вам необходимо обновить драйвер Libcloud, дабы использовать Amazon EC2 Driver (<http://mng.bz/wo95>) или Azure ARM Compute Driver (<http://mng.bz/qY9x>).

Используя инфраструктурный API в качестве уровня абстракции, вы учитываете эволюцию сети независимо от сервера. Например, что произойдет, если вы измените диапазон IP-адресов сети? Обновление развертывается в сети перед запуском IaC для сервера. Сервер вызывает API инфраструктуры для получения сетевых атрибутов и обнаруживает новый диапазон IP-адресов. Затем он заново вычисляет пятый IP-адрес. На рис. 4.7 показана реакция сервера на изменения, вызванные внедрением зависимостей. Когда вы меняете диапазон IP-адресов для сети, ваш сервер получает обновленный диапазон адресов и при необходимости перераспределяет IP-адрес.

Благодаря инверсии зависимостей вы можете развивать низкоуровневые ресурсы отдельно от зависимостей. Инверсия управления помогает ресурсам верхнего уровня реагировать на изменения ресурсов нижнего уровня. Объединение этих двух методов путем внедрения зависимостей обеспечивает компонуемость системы, по-

сколько вы можете добавлять больше ресурсов верхнего уровня к ресурсам нижнего уровня. Внедрение зависимостей помогает свести к минимуму вероятность неудачных изменений в модулях вашей системы.

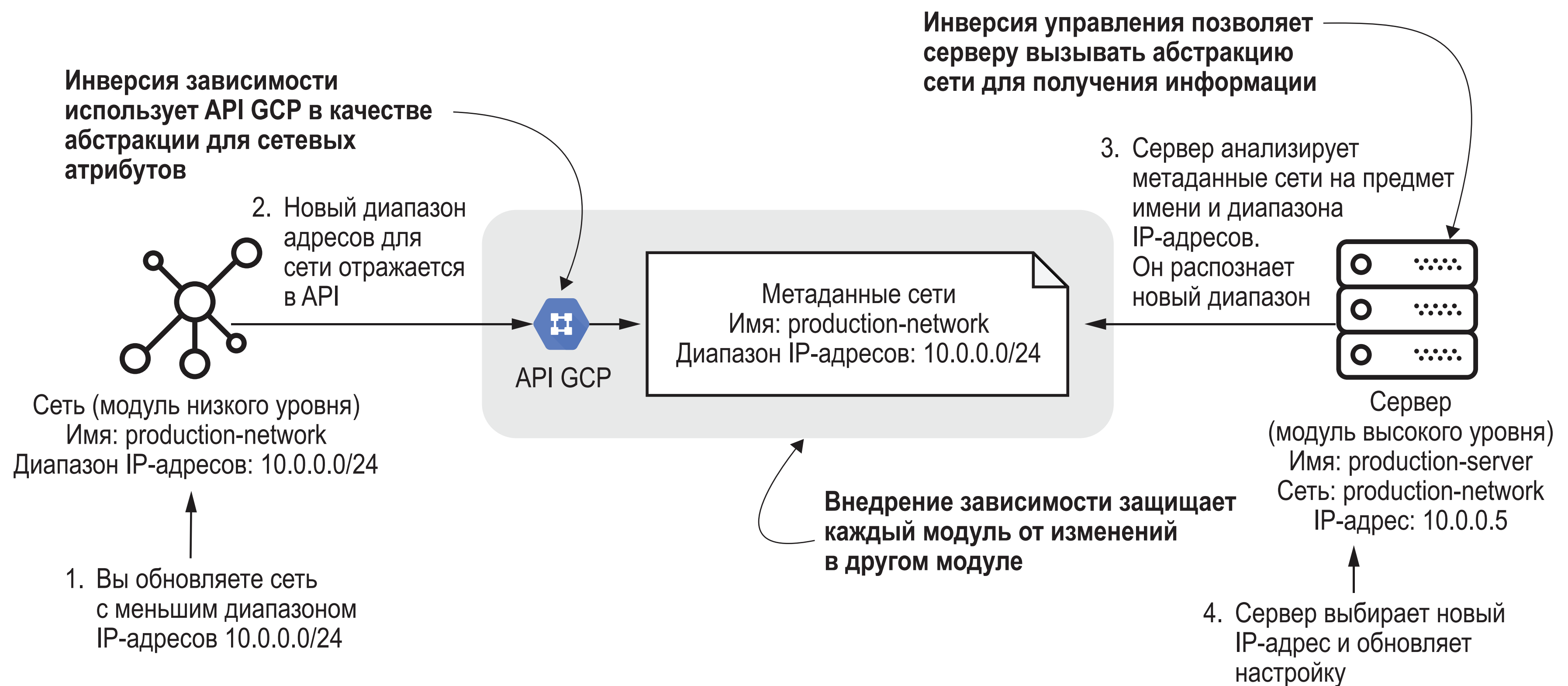


Рис. 4.7 Внедрение зависимостей позволяет изменить модуль нижнего уровня (сеть) и автоматически распространить изменения на модуль верхнего уровня (сервер)

В целом вы должны рассматривать внедрение зависимостей как важнейший принцип управления зависимостями инфраструктуры. Если вы применяете внедрение зависимостей при написании конфигурации инфраструктуры, вы в достаточной степени разделяете (т. е. «рассоединяете») зависимости, чтобы инфраструктурные компоненты можно было изменять независимо друг от друга. По мере роста вашего модуля вы можете продолжать рефакторинг с использованием более специализированных паттернов и еще больше разделять зависимости.

4.3 Паттерн «фасад»

Применение принципа внедрения зависимости порождает аналогичные паттерны для выражения зависимостей. Эти паттерны можно рассматривать как аналог паттернов структурного проектирования при разработке ПО. Стремясь к разделению зависимостей, я часто повторяю в своем IaC одни и те же три паттерна.

Представьте, что вы хотите создать бакет объектного хранилища для хранения статических файлов. Вы можете управлять доступом к файлам с помощью API контроля доступа в GCP. На рис. 4.8 мы создаем сегмент и задаем выходные данные, включающие имя сегмента. Правила контроля доступа для сегмента могут извлекать из выходных данных имя сегмента.

Схема использования выходных данных и уровня абстракции кажется *очень* знакомой. Действительно, вы столкнулись с ней в первой половине главы. Вы, сами того не подозревая, использовали паттерн «фасад» для передачи нескольких атрибутов между модулями!

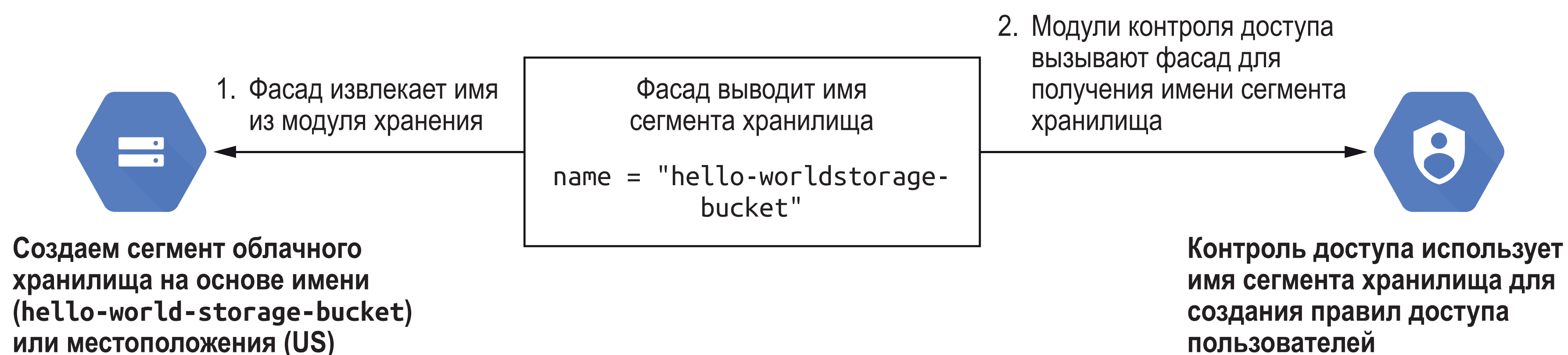


Рис. 4.8 Фасад упрощает атрибуты до имени сегмента хранилища, извлекаемого модулем контроля доступа

Паттерн «фасад» использует выходные данные модуля в качестве абстракции для внедрения зависимостей. Он ведет себя как зеркало, отражая атрибуты других модулей и ресурсов.

ОПРЕДЕЛЕНИЕ Паттерн «фасад» (facade pattern) выводит атрибуты из ресурсов в модуле для внедрения зависимостей.

Фасад лишь отражает атрибуты, и не более того. Паттерн разделяет зависимости между ресурсами высокого и низкого уровней и следует принципу внедрения зависимостей. Ресурс высокого уровня по-прежнему обращается к ресурсу низкого уровня для получения информации, а выходные данные служат абстракцией.

В листинге 4.6 паттерн «фасад» реализован в коде путем создания метода вывода. Модуль сегмента возвращает объект сегмента и имя в своем методе вывода. Модуль доступа использует метод вывода для получения объекта сегмента хранилища и доступа к его имени.

Листинг 4.6 Вывод в фасад имени сегмента для правил контроля доступа

```
import json
import re

class StorageBucketFacade:
    def __init__(self, name):
        self.name = name

class StorageBucketModule:
    def __init__(self, name, location='US'):
        self.name = f'{name}-storage-bucket'
        self.location = location
        self.resources = self._build()

    def _build(self):
```



```

    return {
        'resource': [
            {
                'google_storage_bucket': [{
                    self.name: [{
                        'name': self.name,
                        'location': self.location,
                        'force_destroy': True
                    }]
                }]
            }
        ]
    }

def outputs(self):
    return StorageBucketFacade(self.name)

class StorageBucketAccessModule:
    def __init__(self, bucket, user, role):
        if not self._validate_user(user):
            print("Please enter valid user or group ID")
            exit()
        if not self._validate_role(role):
            print("Please enter valid role")
            exit()
        self.bucket = bucket
        self.user = user
        self.role = role
        self.resources = self._build()

    def _validate_role(self, role):
        valid_roles = ['READER', 'OWNER', 'WRITER']
        if role in valid_roles:
            return True
        return False

    def _validate_user(self, user):
        valid_users_group = ['allUsers', 'allAuthenticatedUsers']
        if user in valid_users_group:
            return True
        regex = r'^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$'
        if re.search(regex, user):
            return True
        return False

    def _change_case(self):
        return re.sub('[^0-9a-zA-Z]+', '_', self.user)

    def _build(self):
        return {
            'resource': [{
                'google_storage_bucket_access_control': [{
                    self._change_case(): [{
                        'bucket': self.bucket.name,

```



```

        'role': self.role,
        'entity': self.user
    }]
    }]
    }]
}

if __name__ == "__main__":
    bucket = StorageBucketModule('hello-world')
    with open('bucket.tf.json', 'w') as outfile:
        json.dump(bucket.resources, outfile, sort_keys=True, indent=4)

    server = StorageBucketAccessModule(
        bucket.outputs(), 'allAuthenticatedUsers', 'READER')
    with open('bucket_access.tf.json', 'w') as outfile:
        json.dump(server.resources, outfile, sort_keys=True, indent=4)

```

- ❶ Используя паттерн «фасад», получаем имя сегмента как часть выходных данных объекта хранилища. Так мы реализуем инверсию зависимостей для абстрагирования от ненужных атрибутов сегмента.
- ❷ Создаем низкоуровневый модуль для сегмента хранилища GCP, который, в свою очередь, использует фабричный паттерн для создания сегмента.
- ❸ Создаем сегмент хранилища Google с использованием ресурса Terraform на основе имени и местоположения.
- ❹ Устанавливаем атрибут в сегменте хранилища Google, чтобы уничтожить его при удалении ресурсов Terraform.
- ❺ Создаем метод вывода для модуля, который возвращает список атрибутов сегмента хранилища.
- ❻ Создаем модуль высокого уровня для добавления правил контроля доступа в сегмент хранилища.
- ❼ Передаем выход фасада сегмента в модуль высокого уровня.
- ❽ Проверяем, что пользователи, переданные в модуль, соответствуют допустимым типам групп пользователей для всех пользователей или всех аутентифицированных пользователей.
- ❾ Проверяем, что роли, переданные в модуль, соответствуют допустимым ролям в GCP.
- ❿ Создаем правила контроля доступа к хранилищу Google с помощью ресурса Terraform.

Эквиваленты AWS и Azure

Сегмент хранилища GCP аналогичен бакету Amazon Simple Storage Service (S3) или BLOB-хранилищу Azure.

Зачем выводить весь объект сегмента, а не только имя? Помните, что нам нужно создать уровень абстракции, соответствующий принципу инверсии зависимостей. Если вы создаете новый модуль, который зависит от местоположения сегмента, вы можете обновить фасад объекта сегмента, чтобы вывести имя и местоположение. Обновление не затрагивает модуль доступа.

Реализация фасада не составляет труда, зато дает преимущества от разделения зависимостей. Одним из таких преимуществ является возможность создания изолированных автономных обновлений в одном модуле, не затрагивая других. Добавление новых высокоуровневых зависимостей не требует особых усилий.

Паттерн «фасад» также упрощает поиск и устранение проблем. Он отражает выходные данные без добавления логики для анализа, что упрощает отслеживание источника проблемы и исправление системы. Подробнее об отмене неудачных изменений вы узнаете в главе 11.

Предметно-ориентированные языки

Вы можете имитировать фасад с помощью DSL, используя выходную переменную с настраиваемым именем. Ресурс высокого уровня ссылается на настроенные выходные имена.

Обычно фасад начинается с одного или двух полей. Всегда придерживайтесь минимального количества полей, необходимых для ресурсов высокого уровня. Просматривайте и удаляйте поля, которые больше не нужны. Рекомендую делать это каждые несколько недель.

Паттерн «фасад» предназначен для более простых зависимостей, таких как зависимость нескольких модулей высокого уровня от одного модуля низкого уровня. Но когда вы добавите много модулей высокого уровня и глубина ваших зависимостей увеличится, у вас возникнут трудности с поддержкой паттерна «фасад» для модулей низкого уровня. Если вам нужно изменить имя поля в выходных данных, вы должны изменить каждый модуль, который ссылается на него. Изменение каждой ссылки на модуль превращается в трудоемкую задачу, если у вас есть сотни ресурсов, которые зависят от одного модуля низкого уровня.

4.4 Паттерн «адаптер»

Фасад отражает значения выходных данных одного модуля инфраструктуры для модулей высокого уровня, о чем говорилось в предыдущем разделе. Он хорошо подходит для простых отношений зависимости, но не работает с более сложными модулями. Более сложные модули обычно включают зависимости «один ко многим» или охватывают нескольких поставщиков инфраструктуры.

Допустим, у вас есть модуль идентификации, который передает список пользователей и ролей для настройки инфраструктуры. Модуль идентификации должен работать на нескольких платформах. На рис. 4.9 показано, что вы настраиваете модуль для вывода

объекта в формате JSON, который сопоставляет такие разрешения, как `read`, `write` или `admin`, с соответствующими именами пользователей. Команды разработчиков должны сопоставить эти имена пользователей и их общие разрешения с условиями, специфичными для GCP. Система доступа GCP использует роли `viewer`, `editor` и `owner`, которые необходимо преобразовать в `read`, `write` и `admin` соответственно.

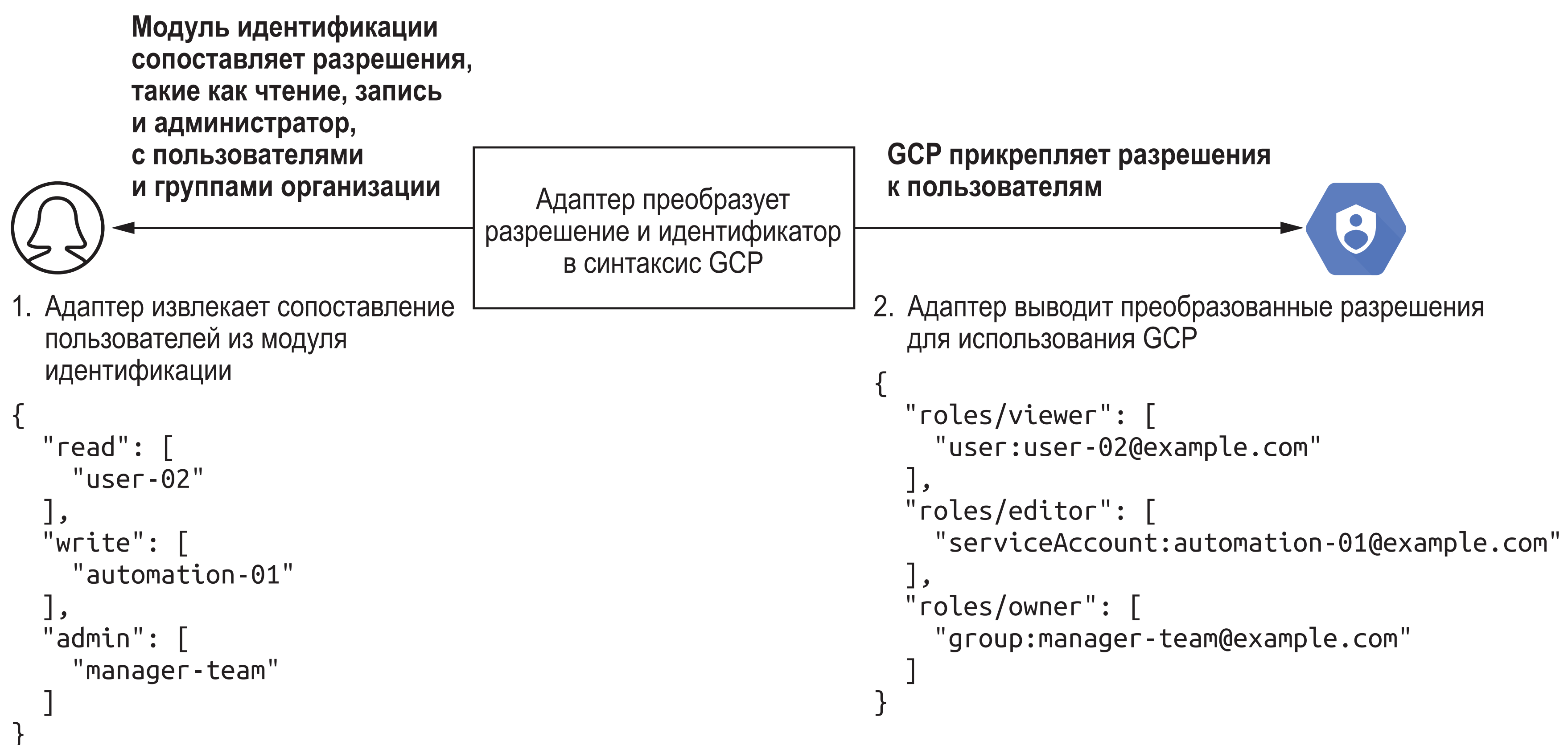


Рис. 4.9 Паттерн «адаптер» преобразует атрибуты в другой интерфейс, который могут использовать модули высокого уровня

Как сопоставить общий набор ролей с конкретными ролями поставщика инфраструктуры? Сопоставление должно гарантировать возможность воспроизведения и развития модуля на платформе любого провайдера инфраструктуры. В будущем вам понадобится расширить модуль, чтобы добавлять пользователей к эквивалентным ролям на разных платформах.

Решением служит *паттерн «адаптер»* (adapter pattern), который преобразует метаданные из ресурса низкого уровня, чтобы их мог использовать любой ресурс высокого уровня. Он похож на электрическую вилку-адаптер для туристов. Вы можете поменять вилку в зависимости от розетки в стране, которую вы посетили, и продолжать пользоваться своими электронными устройствами.

ОПРЕДЕЛЕНИЕ Паттерн «адаптер» извлекает и преобразовывает метаданные из ресурса или модуля низкого уровня, чтобы их мог использовать любой ресурс или модуль высокого уровня.

Для начала мы создаем словарь, который сопоставляет общие имена ролей с пользователями. В листинге 4.7 мы назначаем группе аудита и двум пользователям роль «только для чтения». Эти общие

роли и имена пользователей не соответствуют ни одному из разрешений и ролей GCP.

Листинг 4.7 Создание статического объекта, который сопоставляет общие роли с именами пользователей

```
class Infrastructure:
    def __init__(self):
        self.resources = {
            'read': [
                'audit-team',
                'user-01',
                'user-02'
            ],
            'write': [
                'infrastructure-team',
                'user-03',
                'automation-01'
            ],
            'admin': [
                'manager-team'
            ]
        }
```

- ① Назначаем команде аудита audit-team, пользователям user-01 и user-02 роль только для чтения. Сопоставление описывает, что пользователь может только читать информацию о любом поставщике инфраструктуры.
- ② Назначаем команде поддержки инфраструктуры infrastructure-team, пользователям user-03 и automation-01 роль с возможностью записи. Сопоставление описывает, что пользователь может обновлять информацию о любом поставщике инфраструктуры.
- ③ Назначаем команде менеджеров роль администратора. Сопоставление описывает, что пользователь может управлять любым поставщиком инфраструктуры.

Эквиваленты AWS и Azure

Для тех, кто лучше знаком с AWS, эквивалентными политиками для каждого набора разрешений будут AdministratorAccess для admin, PowerUserAccess для write и ViewOnlyAccess для read. Управление доступом на основе ролей Azure использует Owner для admin, Contributor для write и Reader для read.

Но в сопоставлениях ролей вы не можете ничего сделать со статическим объектом. GCP не понимает имена пользователей и роли! Создайте паттерн «адаптер», чтобы сопоставить общие разрешения с разрешениями, специфичными для инфраструктуры.

В листинге 4.8 мы создаем адаптер идентификации, специфичный для GCP, который сопоставляет общие разрешения, такие как read, с терминами, специфичными для GCP, такими как roles/viewer. GCP может использовать отображение для добавления пользователей, учетных записей служб и групп к правильным ролям.

Листинг 4.8 Использование паттерна «адаптер» для преобразования общих разрешений

```

import json
import access

class GCPIIdentityAdapter:
    EMAIL_DOMAIN = 'example.com'

    def __init__(self, metadata):
        gcp_roles = {
            'read': 'roles/viewer',
            'write': 'roles/editor',
            'admin': 'roles/owner'
        }
        self.gcp_users = []
        for permission, users in metadata.items():
            for user in users:
                self.gcp_users.append(
                    (user, self._get_gcp_identity(user),
                     gcp_roles.get(permission)))

    def _get_gcp_identity(self, user):
        if 'team' in user:
            return f'group:{user}@{self.EMAIL_DOMAIN}'
        elif 'automation' in user:
            return f'serviceAccount:{user}@{self.EMAIL_DOMAIN}'
        else:
            return f'user:{user}@{self.EMAIL_DOMAIN}'

    def outputs(self):
        return self.gcp_users

class GCPProjectUsers:

    def __init__(self, project, users):
        self._project = project
        self._users = users
        self.resources = self._build()

    def _build(self):
        resources = []
        for (user, member, role) in self._users:
            resources.append({
                'google_project_iam_member': [{
                    user: [{
                        'role': role,
                        'member': member,
                        'project': self._project
                    }]
                }]
            })
        return {
            'resource': resources

```



```

    }

if __name__ == "__main__":
    users = GCPIIdentityAdapter(
        access.Infrastructure().resources).outputs() 14

    with open('main.tf.json', 'w') as outfile: 15
        json.dump( 15
            GCPPProjectUsers( 15
                'infrastructure-as-code-book', 15
                users).resources, outfile, sort_keys=True, indent=4) 15

```

- ❶ Создаем адаптер для сопоставления общих типов ролей с типами ролей Google.
- ❷ Задаем домен электронной почты как константу, которую будем добавлять к каждому пользователю.
- ❸ Создаем словарь для сопоставления общих ролей с разрешениями и ролями, специфичными для GCP.
- ❹ Для каждого разрешения и пользователя создаем кортеж с пользователем, идентификатором GCP и ролью.
- ❺ Преобразуем имена пользователей в термины, специфичные для GCP, в состав которых входят тип пользователя и адрес электронной почты.
- ❻ Если в имени пользователя есть слово team, идентификатор GCP должен иметь префикс group- и суффикс в виде домена электронной почты.
- ❼ Если в имени пользователя есть слово automation, идентификатор GCP должен иметь префикс serviceAccount- и суффикс в виде домена электронной почты.
- ❽ Для всех остальных пользователей идентификатор GCP должен иметь префикс user- и суффикс в виде домена электронной почты.
- ❾ Выводим список кортежей, содержащих пользователей, идентификаторы GCP и роли.
- ❿ Создаем модуль для пользователей проекта GCP, использующий фабричный паттерн для прикрепления пользователей к ролям GCP для данного проекта.
- ⓫ Применяем модуль для создания конфигурации JSON пользователей и ролей проекта.
- ⓬ Создаем словарь для сопоставления общих ролей с разрешениями и ролями, специфичными для GCP.
- ⓭ Создаем список участников IAM проекта Google, используя ресурс Terraform. Список извлекает идентификатор, роль и проект GCP, чтобы связать имя пользователя с правами чтения, записи или администратора в GCP.
- ⓮ Создаем адаптер для сопоставления общих типов ролей с типами ролей Google.
- ⓯ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Чтобы перенести этот код в AWS, вы должны сопоставить ссылки на проект GCP с учетной записью AWS. Пользователи проекта GCP соответствуют пользователю AWS IAM и связанным с ним ролям. Аналогичным образом создают подписку Azure и добавляют учетную запись пользователя и его разрешения API в Azure Active Directory.

Вы можете расширить свой адаптер идентификации, чтобы сопоставить общий словарь требований доступа с другим поставщиком инфраструктуры, например AWS или Azure. Как правило, адаптер переводит язык, специфичный для конкретного провайдера или модуля-прототипа, в общие термины. Этот паттерн лучше всего работает для модулей с разными поставщиками инфраструктуры или зависимостями. Я также использую паттерн «адаптер» для создания согласованного интерфейса с провайдерами инфраструктуры в случае плохо определенных параметров ресурсов.

В качестве более сложного примера представьте настройку соединения виртуальной частной сети (VPN) между двумя облаками. Вместо передачи сетевой информации от каждого провайдера через фасад вы используете адаптер, как показано на рис. 4.10. Ваши сетевые модули для каждого провайдера выводят сетевой объект с более общими полями, такими как имя и IP-адрес. В этой ситуации выгоден именно адаптер, поскольку он согласовывает семантику двух разных языков (например, шлюза GCP Cloud VPN и шлюза клиента AWS).

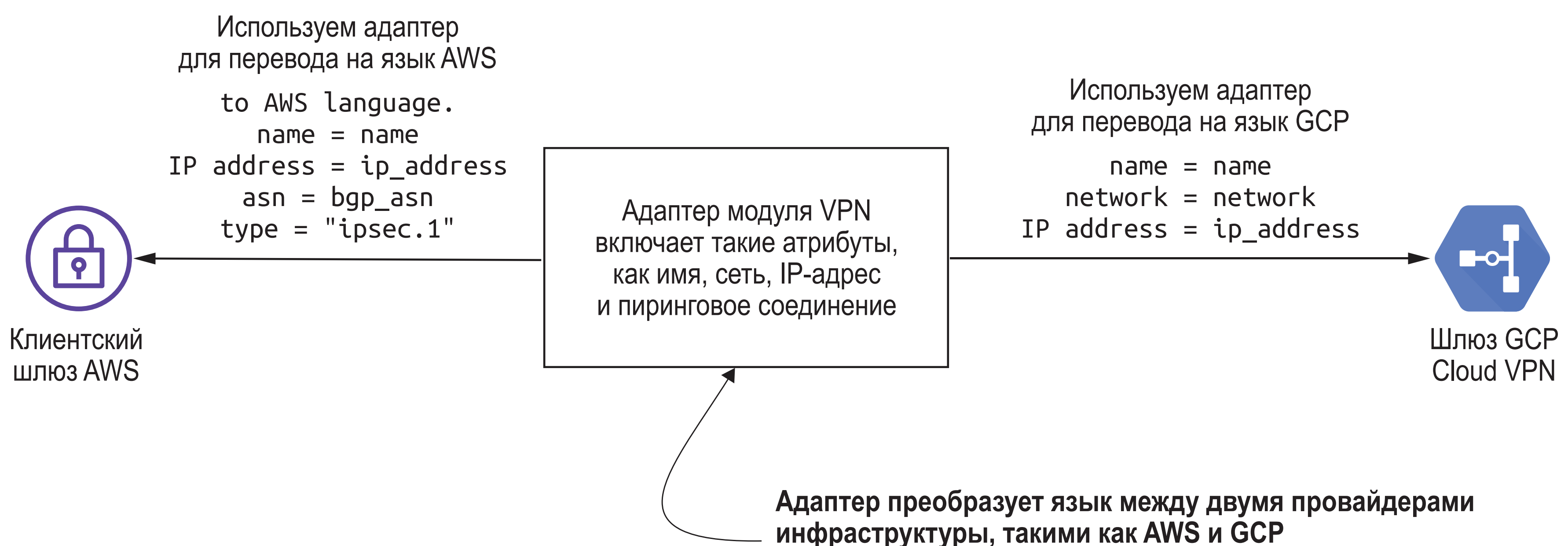


Рис. 4.10 Адаптер переводит язык и атрибуты между двумя поставщиками облачных услуг

Эквивалент Azure

VPN-шлюз Azure обеспечивает функциональность, аналогичную клиентскому шлюзу AWS и шлюзу GCP Cloud VPN.

Как применение адаптера способствует улучшению компонентности и развиваемости? Этот паттерн в значительной степени опирается на инверсию зависимостей, чтобы абстрагировать любое преобразование атрибутов между ресурсами. Адаптер ведет себя как контракт между модулями. Пока оба модуля согласны с контрактом, изложенным адаптером, вы можете продолжать изменять модули высокого и низкого уровней почти независимо друг от друга.

Предметно-ориентированные языки

DSL реализуют адаптер в своем фреймворке для представления состояния инфраструктуры. Состояние инфраструктуры часто включает в себя те же метаданные ресурса, что и API инфраструктуры. Некоторые инструменты позволяют вам взаимодействовать с файлом состояния и рассматривать схему как адаптер для модулей высокого уровня.

Но паттерн «адаптер» работает только в том случае, если вы *соблюдаете контракт* между модулями. Напомню, что мы создали адаптер для преобразования разрешений и имен пользователей в GCP. Что произойдет, если ваш коллега по команде случайно сопоставит роль только для чтения с разрешением `roles/reader`, которого не существует? На рис. 4.11 показано, что если вы не примените правильную роль, специфичную для GCP, произойдет сбой сценария IaC.

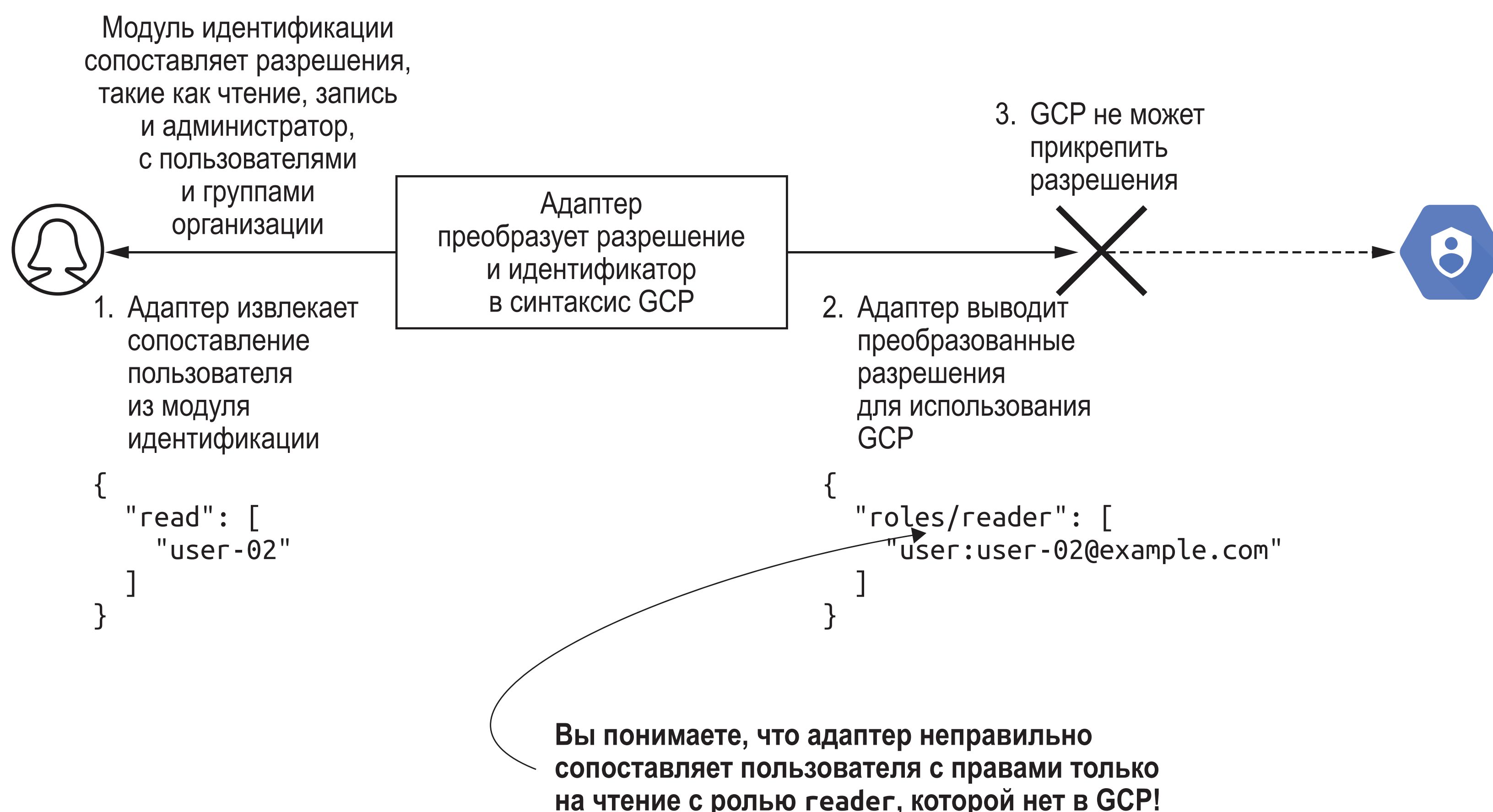


Рис. 4.11 Вам необходимо устранить неполадки и протестировать адаптер, чтобы правильно сопоставить поля

В этом примере вы нарушили контракт между универсальной ролью и ролями GCP! Нарушение контракта приводит к сбою вашего сценария IaC. Убедитесь, что вы поддерживаете и обновляете правильные сопоставления в своем адаптере, чтобы свести к минимуму вероятность сбоя.

Кроме того, наличие адаптера затрудняет устранение неполадок. Паттерн усугубляет путаницу ресурсов, зависящих от конкретного атрибута адаптера. Вам необходимо выяснить, является ли ошибка результатом неправильного вывода поля из исходного модуля, неправильного атрибута в адаптере или неправильного поля, исполь-

зуемого зависимым модулем. Механизмы управления версиями модулей и тестирования, описанные в главах 5 и 6 соответственно, помогают найти и устранить неполадки адаптера.

4.5 Паттерн «медиатор»

Паттерны «адаптер» и «фасад» изолируют изменения и упрощают управление одной зависимостью. Однако IaC часто описывает сложные зависимости ресурсов. Чтобы распутать паутину зависимостей, вы можете создать продуманный сценарий автоматизации, который в четкой структурированной форме описывает, когда и как будут создаваться ресурсы.

Представьте, что вы хотите добавить правило брандмауэра, разрешающее SSH доступ к IP-адресу сервера в нашем каноническом примере сервера и сети. Но вы можете создать правило брандмауэра, только если сервер существует. Аналогично вы можете создать сервер только в том случае, если сеть существует. Вам требуется сценарий автоматизации, способный охватить всю сложность взаимоотношений между межсетевым экраном, сервером и сетью.

Попробуем уловить логику создания сети, сервера и брандмауэра. Хороший сценарий автоматизации способен определить, какие ресурсы создавать в первую очередь. На рис. 4.12 показан рабочий процесс автоматизации. Если ресурс является сервером, IaC создает сеть, а затем сервер. Если ресурс является правилом брандмауэра, IaC сначала создает сеть, затем сервер, а потом правило брандмауэра.

IaC реализует внедрение зависимостей для абстрагирования и управления зависимостями сети, сервера и брандмауэра. Он основан на принципе идемпотентности для непрерывной работы и достижения одного и того же конечного состояния (сеть, сервер и брандмауэр) независимо от существующих ресурсов. Хорошая компонентность облегчает создание компонентов инфраструктурных ресурсов и зависимостей.

Паттерн «медиатор» работает как служба управления воздушным движением в аэропорту. Он контролирует ситуацию и управляет прибывающими и исходящими рейсами. Единственная цель медиатора – организовать зависимости между этими ресурсами и создавать или удалять объекты по мере необходимости.

ОПРЕДЕЛЕНИЕ Паттерн «медиатор» (mediator pattern) организует зависимости между ресурсами инфраструктуры и реализует логику для создания или удаления объектов на основе их зависимостей.

Рассмотрим пример паттерна «медиатор» для сети, сервера и брандмауэра. Реализация медиатора в Python требует нескольких операторов `if-else` для проверки каждого типа ресурса и построе-

ния его низкоуровневых зависимостей. В листинге 4.9 создание брандмауэра зависит от наличия сервера и сети.

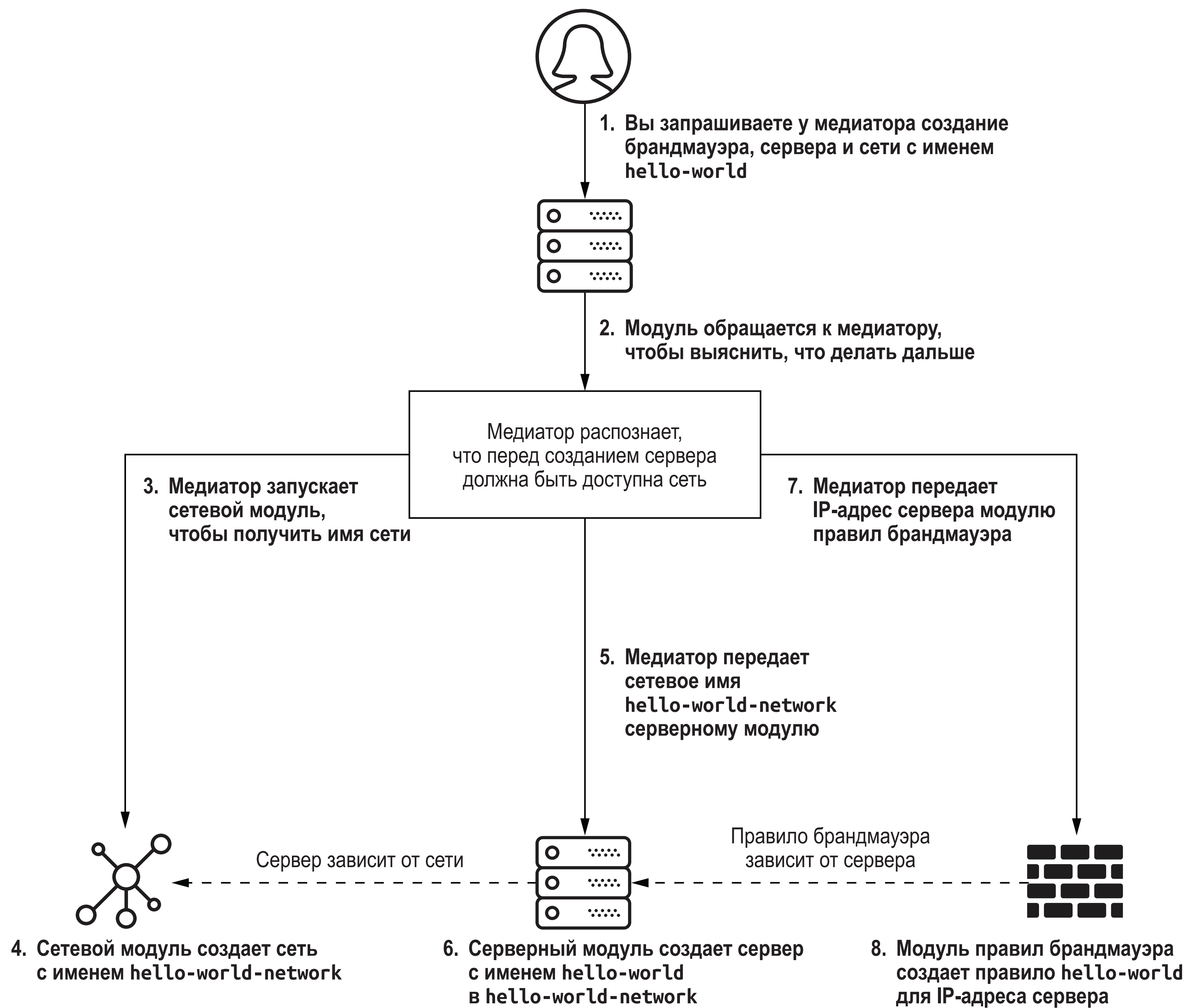


Рис. 4.12 Медиатор отвечает за решение, какой ресурс настраивать в первую очередь

Листинг 4.9 Использование паттерна «медиатор» для организации сервера и зависимостей

```
import json
from server import ServerFactoryModule
from firewall import FirewallFactoryModule
from network import NetworkFactoryModule

class Mediator:
    def __init__(self, resource, **attributes):
        self.resources = self._create(resource, **attributes)

    def _create(self, resource, **attributes):
        if isinstance(resource, FirewallFactoryModule):
            server = ServerFactoryModule(resource._name)
            resources = self._create(server)
```



```

        firewall = FirewallFactoryModule(
            resource._name, depends_on=resources[1].outputs())
        resources.append(firewall)
    elif isinstance(resource, ServerFactoryModule):
        network = NetworkFactoryModule(resource._name)
        resources = self._create(network)
        server = ServerFactoryModule(
            resource._name, depends_on=network.outputs())
        resources.append(server)
    else:
        resources = [resource]
    return resources

def build(self):
    metadata = []
    for resource in self.resources:
        metadata += resource.build()
    return {'resource': metadata}

if __name__ == "__main__":
    name = 'hello-world'
    resource = FirewallFactoryModule(name)
    mediator = Mediator(resource)

    with open('main.tf.json', 'w') as outfile:
        json.dump(mediator.build(), outfile, sort_keys=True, indent=4)

```

- ❶ Импортируем модули фабрики для сети, сервера и брандмауэра.
- ❷ Создаем медиатор, который решает, как и в каком порядке автоматизировать изменения в ресурсах.
- ❸ Когда вы применяете медиатор для создания ресурса, такого как сеть, сервер или брандмауэр, вы позволяете ему самостоятельно решить, какие ресурсы нужно настроить.
- ❹ Если вы хотите создать правило брандмауэра, медиатор сначала рекурсивно вызовет себя, чтобы создать сервер.
- ❺ После того как медиатор создаст конфигурацию сервера, он создает конфигурацию правил брандмауэра.
- ❻ Если вы хотите создать сервер, медиатор сначала рекурсивно вызовет себя, чтобы создать сеть.
- ❼ После того как посредник создаст конфигурацию сети, он создает конфигурацию сервера.
- ❽ Если вы передадите медиатору какой-либо другой ресурс, например сеть, он создаст конфигурацию по умолчанию.
- ❾ Применяем модуль для создания списка ресурсов от медиатора и рендеринга конфигурации JSON.
- ❿ Передаем медиатору ресурс межсетевого экрана. Медиатор создаст сеть, сервер, а затем конфигурацию брандмауэра.
- ⓫ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Правила брандмауэра для GCP по поведению аналогичны правилам для группы безопасности AWS или группы безопасности сети Azure. Правила контролируют входящий и исходящий трафики в диапазоне IP-адресов для указанных ресурсов.

Если вам нужен новый ресурс, например балансировщик нагрузки, вы можете доработать медиатор, чтобы построить его после сервера или брандмауэра. Паттерн «медиатор» лучше всего работает с модулями, которые имеют много уровней зависимостей и несколько системных компонентов.

Однако реализация медиатора может оказаться сложной задачей. Паттерн «медиатор» должен следовать принципу идемпотентности. Если вы запустите медиатор несколько раз, система должна достичь одного и того же целевого состояния. Вам придется написать и протестировать всю логику медиатора. Отказавшись от тестирования, вы рискуете вызвать сбой ресурса или всей инфраструктуры. Для разработки собственного медиатора придется написать большой объем кода!

К счастью, вам не придется часто создавать собственный медиатор. Большинство инструментов IaC действуют как медиаторы, разрешая сложные зависимости и решая, как создавать ресурсы. Многие инструменты конфигурирования имеют встроенные медиаторы для определения зависимостей и порядка операций. Например, оркестратор контейнеров Kubernetes использует медиатор для оркестрации изменений ресурсов в кластере. Ansible использует медиатор, чтобы определить, какие шаги автоматизации следует объединить в композицию и запустить из различных модулей конфигурации.

ПРИМЕЧАНИЕ Некоторые инструменты IaC реализуют паттерн «медиатор», используя теорию графов для отображения зависимостей между ресурсами. Ресурсы служат узлами графа, а его ребра передают атрибуты зависимым ресурсам. Если вы хотите создавать ресурсы без инструмента, попробуйте вручную составить диаграмму зависимостей в вашей системе. Диаграммы могут облегчить разработку сценария автоматизации и кода. На них также хорошо видно, зависимости между какими модулями можно отделить. Построение графов зависимостей помогает реализовать медиатор.

Я создаю свой паттерн «медиатор» только тогда, когда не могу найти его в инструменте или мне нужно решить задачу, для которой требуется больше одного инструмента. Например, я иногда разрабатываю медиатор для управления созданием кластера Kubernetes в одном инструменте до того, как другой инструмент развернет сервисы в кластере Kubernetes. Медиатор согласовывает шаги автоматизации между этими двумя инструментами, например проверяет работоспособность кластера перед развертыванием служб с помощью второго инструмента.

4.6 Выбор паттерна

Фасад, адаптер и медиатор используют внедрение зависимостей для разделения изменений между модулями высокого и низкого уровней. Вы можете применить любой из паттернов, и они будут выражать зависимости между модулями и изолировать изменения внутри них. По мере роста вашей системы вам может потребоваться изменить эти паттерны в зависимости от структуры вашего модуля.

Выбор паттерна определяется количеством зависимостей от низкоуровневого модуля или ресурса. Паттерн «фасад» предназначен для одного модуля низкого уровня и нескольких модулей высокого уровня. Если у вас есть модуль низкого уровня со многими зависимостями модулей высокого уровня, имеет смысл применить адаптер. Если у вас много зависимостей между модулями, вам может понадобиться медиатор для управления автоматизацией ресурсов. На рис. 4.13 показано дерево решений для определения того, какой паттерн зависимостей использовать.

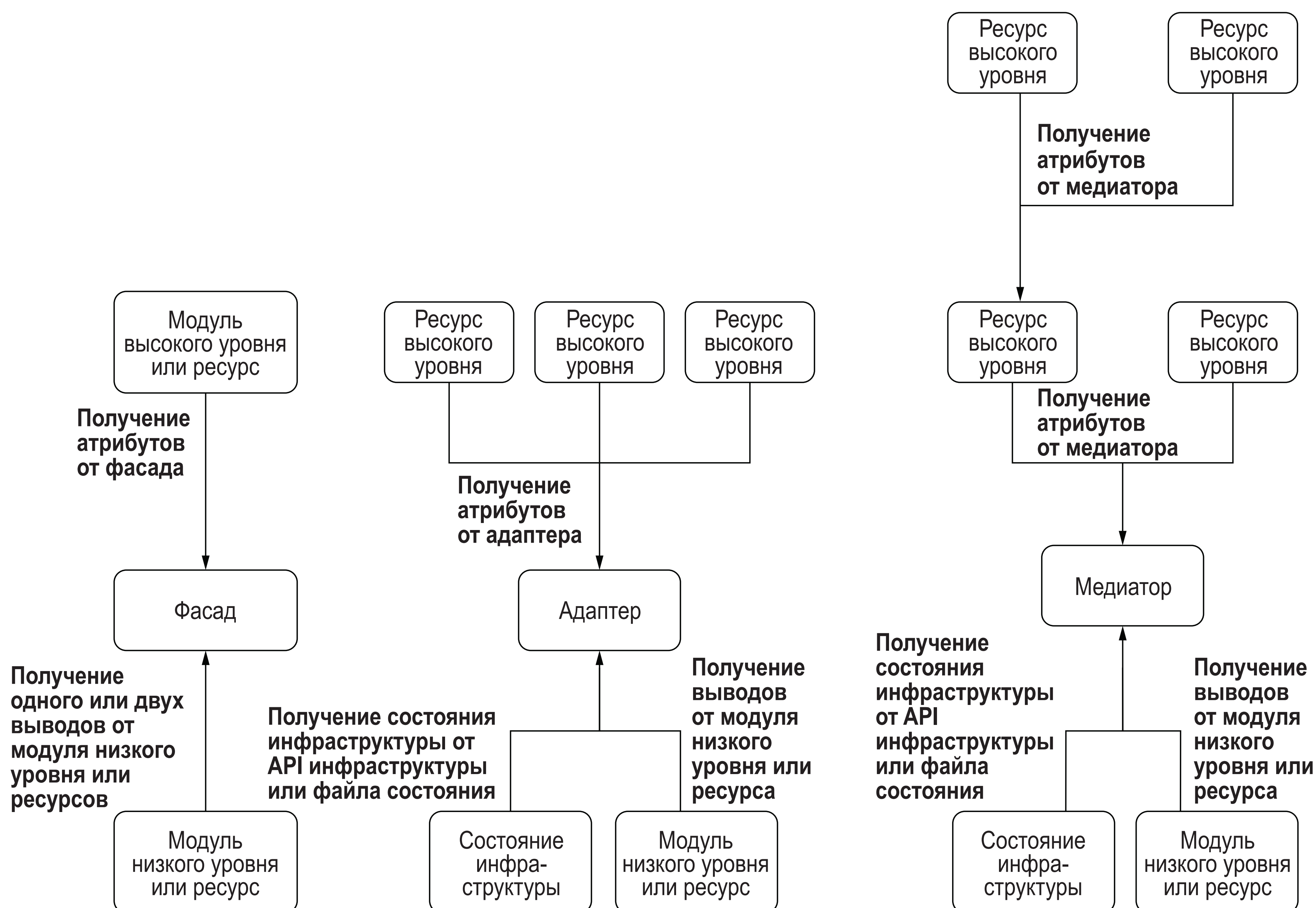
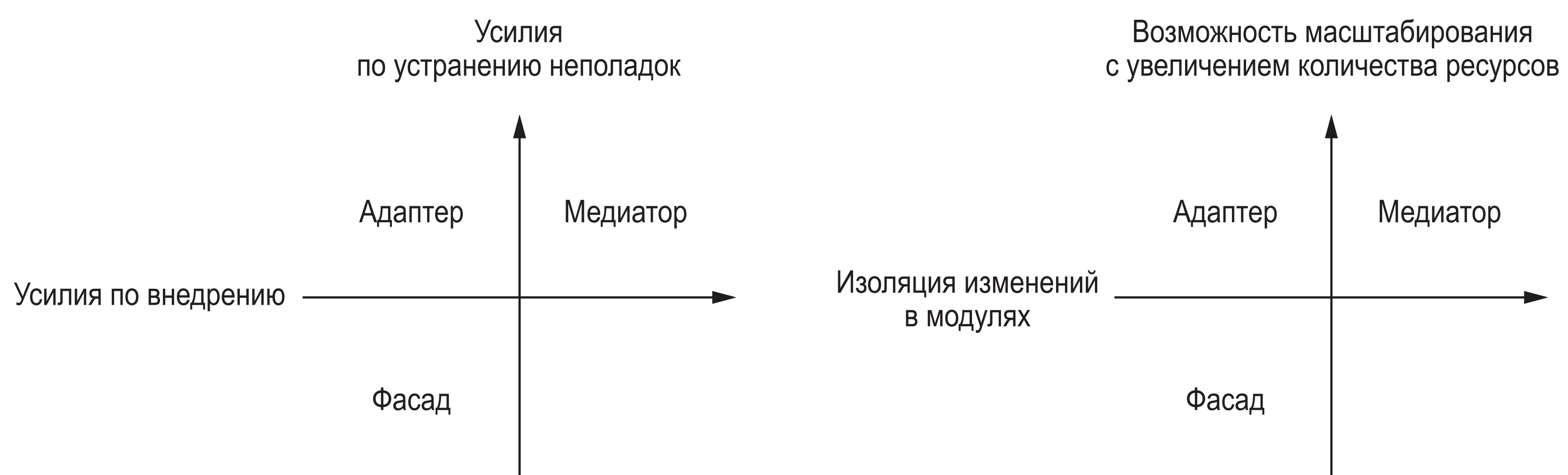


Рис. 4.13 Выбор абстракции зависит от отношения зависимости: внутримодульная, «один к одному» или «один ко многим»

Все паттерны улучшают идемпотентность, композируемость и развиваемость посредством внедрения зависимостей. Но зачем начинать с фасада, а затем рассматривать адаптер или медиатор? По

мере роста вашей системы вам потребуется оптимизировать паттерн управления зависимостями, чтобы уменьшить операционную нагрузку, связанную с изменениями.

На рис. 4.14 показана взаимосвязь между устранением неполадок и усилиями по реализации, а также масштабируемостью и изоляцией для паттернов «фасад», «медиатор» и «адаптер». Например, преимущество фасада заключается в минимальных усилиях по реализации и устранению неполадок, но он не позволяет масштабировать или изолировать изменения с помощью большего количества ресурсов. Адаптеры и медиаторы обеспечивают улучшенную масштабируемость и изоляцию ценой возросших трудозатрат по устранению неполадок и внедрению.



Хотя шаблон адаптера может требовать высоких затрат, связанных с устранением неполадок и внедрением, он обеспечивает лучшую масштабируемость с большим количеством ресурсов и изоляцией изменений в модулях

Рис. 4.14 Некоторые паттерны могут иметь низкую стоимость устранения неполадок и реализации, но не могут изолировать изменения в модулях и масштабироваться

Сократите первоначальные усилия, выбрав инструмент со встроенным медиатором. Затем используйте встроенную в инструмент реализацию фасада для управления зависимостями между модулями или ресурсами. Если вам сложно управлять фасадом из-за того, что у вас несколько систем, зависящих друг от друга, попробуйте изучить и применить адаптер или медиатор.

Адаптер требует больше усилий для реализации, но обеспечивает лучшую основу для расширения и роста вашей инфраструктурной системы. Вы всегда можете добавлять новых провайдеров инфраструктуры и системы, не беспокоясь об изменении модулей низкого уровня. Но вы не можете рассчитывать на использование адаптера для каждого модуля, поскольку для реализации и устранения неполадок требуется время.

Инструмент с медиатором сам выбирает, какие компоненты будут обновляться и когда. Готовый инструмент снижает общие усилия по внедрению, но создает некоторые проблемы при устранении неполадок. Вам необходимо знать поведение вашего инструмента, чтобы устранять неудачные изменения зависимостей. Иногда может получиться так, что инструмент с медиатором облегчит масштабирование, но не сможет полностью изолировать изменения в модулях.

Упражнение 4.1

Каким наилучшим способом можно смягчить зависимость базы данных от сети с помощью следующего кода IaC:

```
class Database:
    def __init__(self, name):
        spec = {
            'name': name,
            'settings': {
                'ip_configuration': {
                    'private_network': 'default'
                }
            }
        }
```

- a** этот подход достаточно хорошо смягчает зависимость базы данных от сети;
- b** нужно передать идентификатор сети как переменную, а не жестко программировать его по умолчанию;
- c** реализуйте и передайте объект NetworkOutput в модуль базы данных для всех сетевых атрибутов;
- d** добавьте в сетевой модуль функцию для передачи его сетевого идентификатора в модуль базы данных;
- e** добавьте в модуль базы данных функцию для получения через API инфраструктуры идентификатора сети по умолчанию.

Ответы на упражнения даны в приложении В.

Краткое содержание главы

- Применяйте паттерны зависимостей инфраструктуры, такие как фасад, адаптер и медиатор, чтобы отделить модули и ресурсы, и вы сможете вносить изолированные изменения в модули.
- Инверсия управления означает, что ресурс высокого уровня обращается к ресурсу низкого уровня для получения атрибутов.
- Принцип инверсии зависимостей гласит, что ресурс высокого уровня должен использовать абстракцию метаданных ресурса низкого уровня.
- Внедрение зависимостей сочетает в себе принципы инверсии управления и инверсии зависимостей.
- Если сразу не ясно, какой паттерн выбрать, воспользуйтесь внедрением зависимостей для ресурса высокого уровня, чтобы вызвать ресурс низкого уровня и проанализировать его объектную структуру на предмет необходимых ему значений.
- Используйте паттерн «фасад» для создания упрощенного интерфейса атрибутов.

- Используйте паттерн «адаптер» для преобразования метаданных одного ресурса во входной формат другого ресурса. Этот паттерн лучше всего работает с ресурсами разных провайдеров инфраструктуры или модулями-прототипами.
- Паттерн «медиатор» организует зависимости между этими ресурсами и создает или удаляет объекты по мере необходимости. Большинство инструментов могут выступать в роли медиатора между ресурсами.

Часть II

Масштабирование для работы в команде

После того как вы изучите методы и паттерны описания инфраструктуры в виде кода, вам наверняка захочется поделиться ими со своей командой. В этой части обсуждаются соображения и рекомендации по совместному использованию инструментов и командной работе над IaC. Согласование общих паттернов между несколькими командами сводит к минимуму конфликты и потенциальные неудачи.

В главе 5 описывается, как структурировать, разделять на версии и выпускать модули, чтобы каждый член команды мог безопасно их обновлять. В главе 6 вы узнаете о стратегиях тестирования IaC. Эти тесты помогут вам проверить стабильность изменений перед их инициализацией в промышленной среде.

Мы добавим тестирование в конвейеры поставки в главе 7, где описываются модели разработки и развертывания. Глава содержит рекомендации по работе с ветвлениями (или их отсутствием!) и безопасной инициализации изменений в промышленной среде. Наконец, в главе 8 обсуждаются приемы и методы, которые вы можете применить к конвейерам тестирования и доставки IaC, чтобы обеспечить безопасные и совместимые конфигурации.

Структурирование и совместное использование модулей

Основные темы главы:

- создание версий модулей и тегов для изменений инфраструктуры;
- выбор одного репозитория вместо нескольких;
- организация общих модулей инфраструктуры для разных команд;
- выпуск модулей инфраструктуры без изменения критических зависимостей.

До этого момента в книге вы изучали методы и паттерны разработки инфраструктуры в виде кода и разбиения их на группы компонентов инфраструктуры. Вы можете написать наиболее оптимальные конфигурации, но при этом возникнут проблемы с обслуживанием и снижением риска сбоя в ваших системах. Трудности возникают из-за того, что ваша команда не стандартизирует методы совместной работы при обновлении модулей инфраструктуры.

Представьте себе, что компания Datacenter for Veggies¹ начинает с автоматизации операций по выращиванию ароматных трав. При-

¹ Центр обработки данных для производителей зелени. – Прим. перев.

ложения в GCP отслеживают и настраивают оптимальные условия для роста трав. Каждая команда использует одиночный паттерн и создает уникальную конфигурацию инфраструктуры.

Со временем компания Datacenter for Veggies набирает популярность, и ее технологии хотят применить к выращиванию вообще любой зелени. Компания нанимает новую команду разработчиков приложений, специализирующуюся на программном обеспечении для выращивания различных растений: от трав до листовой зелени и корнеплодов. Каждая команда создает конфигурацию инфраструктуры независимо от других.

Datacenter for Veggies продолжает расширять свой бизнес и нанимает вас для разработки приложения по выращиванию фруктов. Вы понимаете, что не можете повторно использовать какие-либо конфигурации инфраструктуры, поскольку они уникальны для каждой овощной команды. Очевидно, что компании нужен последовательный и пригодный для многократного применения способ построения, защиты и управления инфраструктурой.

Вы понимаете, что Datacenter for Veggies может использовать некоторые паттерны из главы 3 для разделения инфраструктуры на модули с возможностью последующей компоновки. Вы рисуете схему, изображенную на рис. 5.1, для организации и координации работы с группами инфраструктуры между несколькими командами. Команды, занимающиеся травами, корнеплодами, листовой зеленью и фруктами, могут использовать стандартизированную конфигурацию сетей, баз данных и серверов.

Совместное использование модулей разными командами способствует воспроизводимости, компоновке и развиваемости. Командам не придется тратить много времени на создание IaC, поскольку они воспроизводят проверенные конфигурации. Члены команды могут выбирать компоновку своих систем и переопределять конфигурации в соответствии со своими конкретными потребностями.

Чтобы полностью реализовать преимущества стандартизированных модулей, вам необходимо учитывать их жизненный цикл разработки, выходящий за рамки регулярных изменений инфраструктуры. В этой главе рассматриваются методы совместного использования и управления инфраструктурными модулями. Вы изучите теорию и практику выпуска стабильных модулей без возникновения критических сбоев в зависимостях более высокого уровня.

5.1 Структура репозитория

Представьте, что каждая команда Datacenter for Veggies использует для своей инфраструктуры одиночный паттерн. Команды Herbs и Leafy Greens понимают, что используют одинаково настроенные

серверы, сети и базы данных. Могут ли они объединить конфигурацию своей инфраструктуры в один модуль?

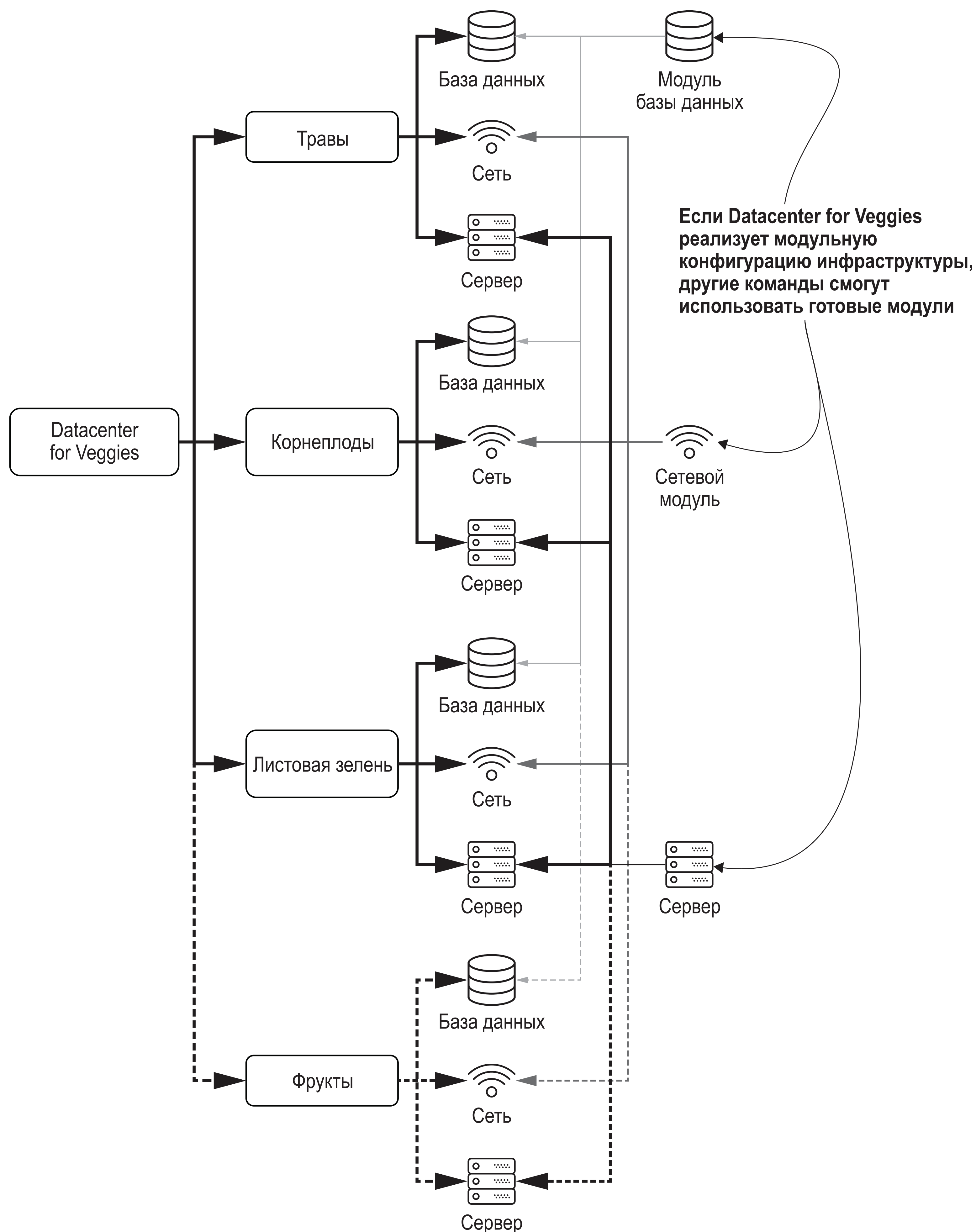


Рис. 5.1 Datacenter for Veggies может использовать модули для организации и стандартизации конфигурации инфраструктуры, которая будет общей для разных команд

Вместо того чтобы заниматься взаимным копированием и вставкой конфигураций, команды Herbs (травы) и Leafy Greens (листовая зелень) хотят разработать единый базовый вариант и ссылаться на него в своей конфигурации. Должен ли Datacenter for Veggies поместить всю инфраструктуру в один репозиторий? Или ему следует разделить свои модули по нескольким репозиториям?

5.1.1 Единый репозиторий

Сначала каждая команда Datacenter for Veggies хранит конфигурацию своей инфраструктуры в едином репозитории кода. Каждая группа помещает свою конфигурацию в специальный каталог, чтобы избежать путаницы конфигураций. Если группа хочет воспользоваться модулем, она импортирует его, указывая локальный путь к файлу.

На рис. 5.2 показано, как Datacenter for Veggies структурирует свой единый репозиторий кода. Репозиторий содержит два каталога верхнего уровня, разделяющих модули и среды. Компания разделяет каталог среды на сегменты для каждой команды, например команды Leafy Greens. В свою очередь, команда Leafy Greens разделяет конфигурации среды разработки и промышленной среды.

При использовании единого репозитория для конфигурирования инфраструктуры разделите модули и конфигурации на разные подкаталоги

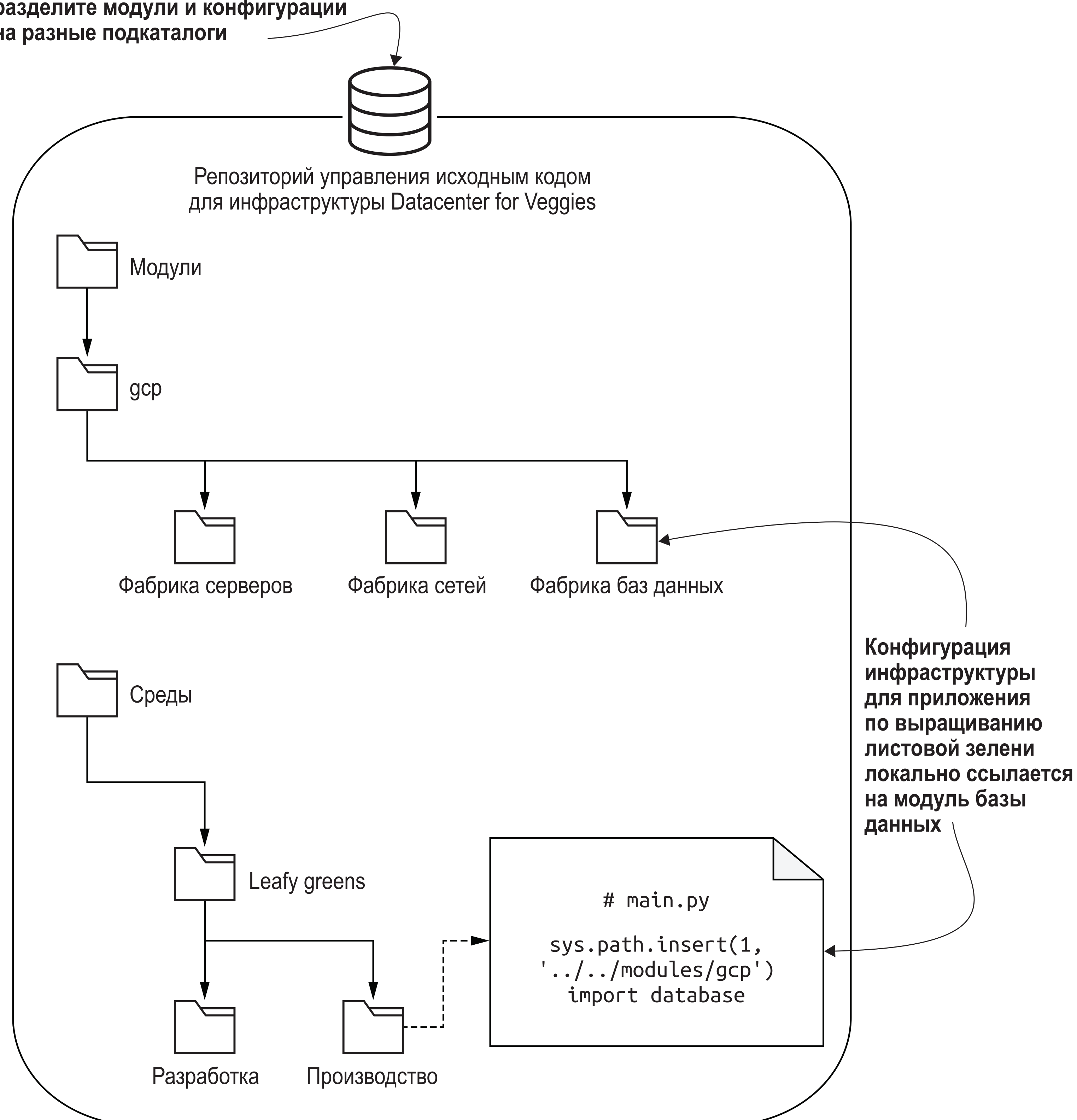


Рис. 5.2 Промышленная среда и среда разработки команды Leafy Greens используют каталоги, содержащие модули сервера, сети и фабрики баз данных, в единой структуре репозитория

Когда члены команды Leafy Greens захотят создать базу данных, они могут использовать готовый модуль из каталога модулей. В своем IaC они импортируют модуль, указывая локальный путь. После импорта они могут использовать фабрику базы данных и создать ресурс в промышленной среде.

Datacenter for Veggies начал определять инфраструктуру с помощью единого репозитория (монорепозитория), в котором будут храниться все конфигурации и модули для каждой команды.

ОПРЕДЕЛЕНИЕ *Единый репозиторий* (или *монорепозиторий*) содержит все IaC (конфигурацию и модули) для команды или функции.

В целом компании нравится единая структура репозитория. Все команды могут воспроизводить свою конфигурацию путем копирования и вставки, а также создавать новые ресурсы, добавляя новый каталог для модуля. В листинге 5.1 члены команды Leafy Greens создают новый модуль базы данных. Используя Python, они вставляют в модули локальный путь к файлу с помощью метода `sys.path`. Они используют базу данных, импортируя ее модуль в кодовую базу.

Листинг 5.1 Ссылка на модули инфраструктуры в другом каталоге

```
import sys
sys.path.insert(1, '.././modules/gcp') ❶

from database import DatabaseFactoryModule ❷
from server import ServerFactoryModule ❷
from network import NetworkFactoryModule ❷

import json

if __name__ == "__main__":
    environment = 'development'
    name = f'{environment}-hello-world'
    network = NetworkFactoryModule(name) ❷
    server = ServerFactoryModule(name, environment, network) ❷
    database = DatabaseFactoryModule(name, server, network, environment) ❷
    resources = { ❸
        'resource': network.build() + server.build() + database.build() ❸
    } ❸

    with open('main.tf.json', 'w') as outfile: ❹
        json.dump(resources, outfile, sort_keys=True, indent=4) ❹
```

- ❶ Импортируем каталог с модулями, поскольку он существует в том же репозитории.
- ❷ Импортируем сервер, базу данных и модули сетевой фабрики для промышленной среды.
- ❸ Применяем модули для создания конфигурации JSON для сети, сервера и базы данных.
- ❹ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Использование локальных папок для хранения модулей помогает командам ссылаться на нужную инфраструктуру. Каждый может искать модули в одном репозитории или изучать конфигурации других команд. Если кто-то из команды Herbs хочет узнать о IaC команды Fruits, он может использовать команду `tree` для изучения структуры каталогов:

```
$ tree .
.
├─ environments
│   ├── fruits
│   │   ├── development
│   │   └─ production
│   ├── herbs
│   │   ├── development
│   │   └─ production
│   └─ leafy-greens
│       ├── development
│       └─ production
└─ roots
    ├── development
    └─ production
```

Чтобы лучше упорядочить конфигурации, каждая группа помещает конфигурации среды разработки и промышленной среды в отдельные каталоги. Эти каталоги изолируют конфигурации и изменения для каждой среды. В идеале все среды должны быть одинаковыми. На самом деле у вас будут различия между средами, связанные с ограничениями затрат или ресурсов.

Другие инструменты

Структура единого репозитория применима ко многим другим инструментам IaC. Вы можете применить структуру единого репозитория для повторного использования ролей и сборников сценариев к таким инструментам управления конфигурацией, как Ansible. У вас будет возможность получать и создавать сборники сценариев или модули управления конфигурацией на основе каждого локального каталога в едином репозитории.

CloudFormation работает немного по-другому. Вы можете разместить все файлы определений стека в едином репозитории. Однако вам необходимо поместить дочерний паттерн (который я считаю модулем) в хранилище S3 и ссылаться на него с помощью параметра `TemplateURL` в ресурсе `AWS::CloudFormation::Stack`. Далее в этой главе вы узнаете, как поставлять и публиковать изменения в модулях.

Datacenter for Veggies использует одного провайдера инфраструктуры – GCP. В будущем команды смогут добавлять новые каталоги

для различных инфраструктурных инструментов или поставщиков. Эти инструменты могут обновлять серверы или сети (каталог `ansible`), создавать образы виртуальных машин (каталог `packer`) или развертывать базу данных на AWS (каталог `aws`):

```
$ tree .
.
├─ environments
│   ├── development
│   └─ production
└─ modules
    ├── ansible
    ├── aws
    ├── gcp
    └─ packer
```

В других публикациях про IaC вы можете столкнуться с принципом «не повторяйся» (DRY, don't repeat yourself). Принцип DRY фокусируется на повторном использовании и компоновке. Модули инфраструктуры уменьшают дублирование и повторение конфигурации, что соответствует подходу DRY. Если у вас могут быть идентичные среды разработки и промышленной среды, вы можете исключить каталоги разработки и промышленной среды и ссылаться на один модуль вместо отдельных файлов для каждой среды.

Вы не сможете полностью соблюдать принцип DRY в инфраструктуре. В зависимости от инфраструктуры или языка и синтаксиса инструмента у вас всегда будет повторяющаяся конфигурация. В результате вам придется время от времени повторять настройки для получения более четкой конфигурации или в рамках ограничений инструментов или платформ.

5.1.2 *Раздельные репозитории*

По мере роста компании Datacenter for Veggies в ее репозитории инфраструктуры накапливаются сотни папок. Каждый каталог содержит множество вложенных папок. Каждую неделю вы тратите время на перебазирование конфигурации со всеми обновлениями из каждого репозитория. А еще вы ждете по 20 минут каждый раз при отправке обновлений в промышленную среду, поскольку ваш фреймворк непрерывной интеграции должен рекурсивно искать изменения. Вдобавок служба безопасности выражает обеспокоенность тем, что подрядчики, работающие с командой Leafy Greens, имеют доступ ко всей инфраструктуре команды Fruits!

Вы разделяете модули сети, тегов, сервера и базы данных на отдельные репозитории. В каждом репозитории есть свой рабочий процесс для создания и доставки модуля, поэтому процедуры непрерывной интеграции занимают меньше времени. Вы можете контролировать доступ к каждому репозиторию, позволяя подрядчикам

команды Leafy Greens получать доступ только к конфигурации Leafy Greens.

Разные команды Datacenter for Veggies могут использовать репозиторий модуля или пакетную версию. Каждая команда хранит свою конфигурацию и модули в отдельном репозитории. Любой сотрудник компании может скачать и использовать модули в своих конфигурациях.

На рис. 5.3 показаны репозитории кода, которые Datacenter for Veggies использует для создания IaC. Каждая команда и модуль получают свой собственный репозиторий кода. Когда команда Leafy Greens хочет создать базу данных, она загружает и импортирует модуль базы данных по URL-адресу репозитория GitHub, а не из локального каталога. Если у команд есть несколько сред, они делят репозиторий кода на каталоги.

При использовании нескольких репозиториях для конфигурации инфраструктуры мы разделяем модули по разным репозиториям

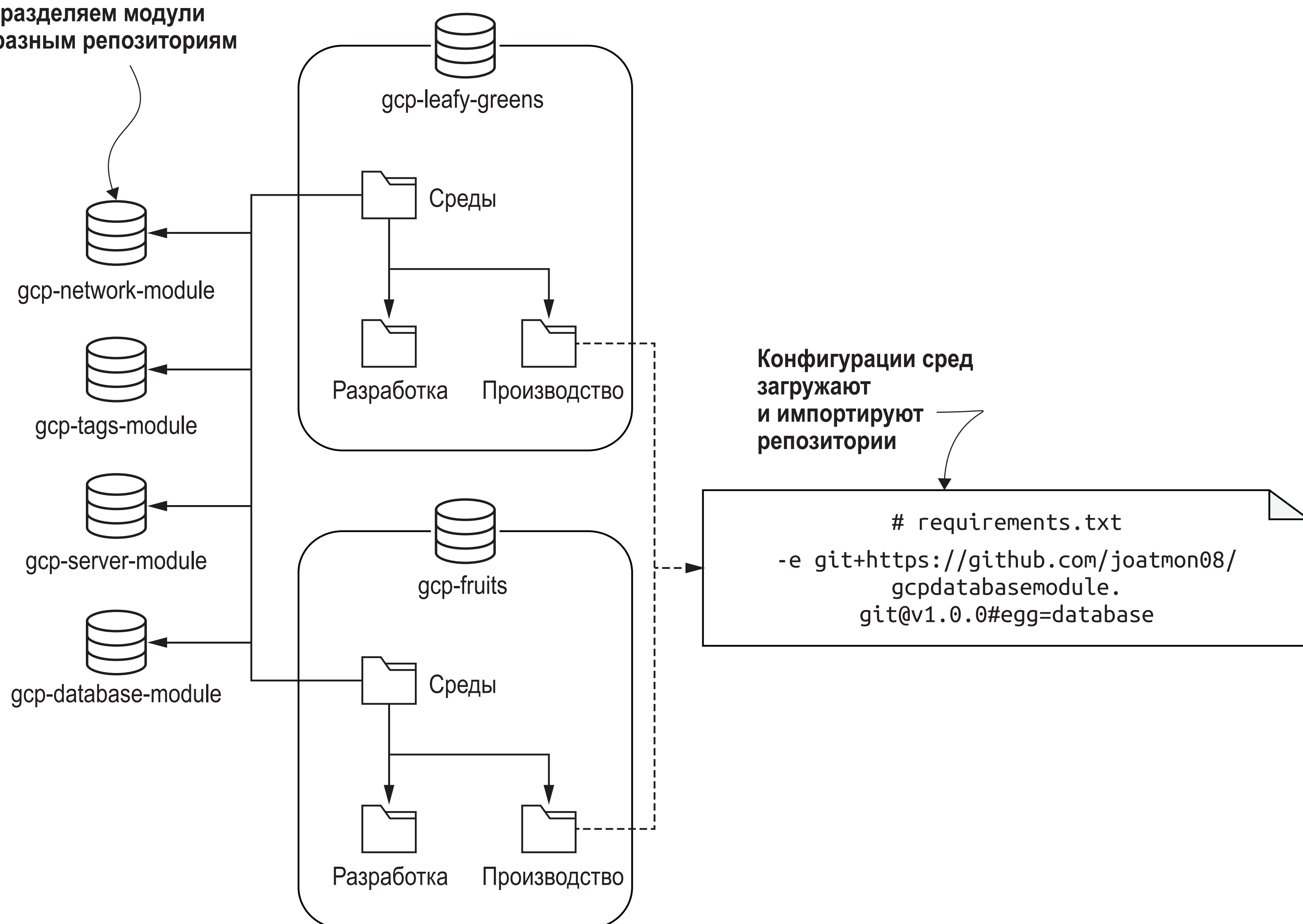


Рис. 5.3 В структуре с отдельными репозиториями каждый модуль хранится в отдельном репозитории кода. Для использования модуля необходимо обратиться по URL-адресу репозитория

Итак, компания Datacenter for Veggies перешла от структуры с единым репозиторием к структуре с отдельными репозиториями. Компания распределила модули в различные репозитории, связанные с конкретными командами.

ОПРЕДЕЛЕНИЕ Структура с *раздельными репозиториями* разделяет IaC (конфигурацию или модули) на разные репозитории в зависимости от команды или функции.

Напомню, что подход с единым репозиторием способствует воспроизводимости и компонуемости. Подход с раздельными репозиториями улучшает развиваемость. Разделение модулей на отдельные репозитории помогает структурировать жизненный цикл и управление каждым модулем.

Чтобы получить структуру с раздельными репозиториями, мы разделяем модули на отдельные репозитории в системе контроля версий. В следующем листинге мы настраиваем менеджер пакетов Python для загрузки каждого модуля, добавляя его в качестве требуемой библиотеки в файле `requirements.txt`. Каждое требование библиотеки должно включать URL-адрес репозитория контроля версий и определенный тег для загрузки.

Листинг 5.2 Ссылки на репозитории модулей в файле `requirements.txt`

```
-e git+https://github.com/joatmon08/gcp-tags-module.git@v1.0.0#egg=tags ❶  
-e git+https://github.com/joatmon08/gcp-network-module.git@v1.0.0#egg=network ❷  
-e git+https://github.com/joatmon08/gcp-server-module.git@v0.0.1#egg=server ❷  
-e git+https://github.com/joatmon08/gcp-database-module.git@v1.0.0#egg=database ❷
```

- ❶ Загружаем модуль-прототип тегов из репозитория GitHub. Выбираем версию модуля на основе тега.
- ❷ Загружаем модуль фабрики для сети, сервера и базы данных из репозитория GitHub. Выбираем версию модуля на основе тега.

Сначала мы создаем репозиторий промышленной конфигурации инфраструктуры для приложения по выращиванию фруктов. После создания репозитория мы добавляем в него файл `requirements.txt`. Затем запускаем менеджер установки пакетов Python, чтобы загрузить каждый модуль конфигурации инфраструктуры:

```
$ pip install -r requirements.txt  
Obtaining tags from  
git+https://github.com/joatmon08/  
➔zgcp-tags-module.git@l.0.0#egg=tags  
...  
Successfully installed database network server tags
```

Вместо того чтобы задавать локальный путь и импортировать модули, нам необходимо сначала запустить менеджер пакетов Python для загрузки из удаленного репозитория. После загрузки модулей команды могут импортировать их в конфигурации среды, используя Python, как показано в листинге 5.3.

Листинг 5.3 Импорт модулей для использования в конфигурации инфраструктуры

```

from tags import StandardTags ❶
from server import ServerFactoryModule ❶
from network import NetworkFactoryModule ❶
from database import DatabaseFactoryModule ❶

import json

if __name__ == "__main__":
    environment = 'development'
    name = f'{environment}-hello-world'

    tags = StandardTags(environment)
    network = NetworkFactoryModule(name)
    server = ServerFactoryModule(name, environment, network, tags.tags)
    database = DatabaseFactoryModule(
        name, server, network, environment, tags.tags)
    resources = {
        'resource': network.build() + server.build() + database.build() ❷
    }

    with open('main.tf.json', 'w') as outfile: ❸
        json.dump(resources, outfile, sort_keys=True, indent=4) ❸

```

- ❶ Импортируем модули, загруженные менеджером пакетов.
- ❷ Применяем модули для создания конфигурации JSON сети, сервера и базы данных.
- ❸ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Как вы знаете, Datacenter for Veggies отдельно настраивает среду разработки и промышленную среду. Команды разрабатывают код, ссылающийся на одни и те же фабрики и модули-прототипы, размещенные в системе контроля версий. Согласованные модули для сред разработки и промышленных сред предотвращают дрейф конфигураций между средами и помогают тестировать изменения модулей перед вводом в эксплуатацию. Вы узнаете больше о тестировании и средах в главе 6.

Реализация IaC для отдельных репозиторий не слишком сильно отличается от одиночного репозитория. Обе структуры поддерживают воспроизводимость и компоновку. Отличие заключается в том, что вы самостоятельно развиваете модуль во внешнем репозитории.

Обновление конфигурации в структуре с отдельными репозиториями подразумевает повторную загрузку новых модулей с помощью вашего менеджера пакетов. Запуск менеджера пакетов для использования нового модуля может привести к сбою рабочего процесса IaC. Кто-то может обновить модуль, но вы не узнаете об этом, пока не просмотрите его репозиторий. Далее в этой главе вы узнаете, как решить эту проблему с помощью системы управления версиями.

Предметно-ориентированные языки

Если инструмент может ссылаться на модули или библиотеки с помощью системы контроля версий или URL-адресов артефактов, значит, он поддерживает структуру с отдельными репозиториями.

Если вы предпочитаете отдельные репозитории, старайтесь придерживаться стандартных приемов совместного использования и обслуживания модулей. Прежде всего стандартизируйте структуру и формат файлов модулей. Это поможет командам вашей организации идентифицировать и фильтровать модули в системе контроля версий. Согласование структуры файлов и продуманное именование репозитория модулей также облегчают аудит и будущую автоматизацию.

Например, модули инфраструктуры в Datacenter for Veggies имеют один и тот же паттерн и файловую структуру. Их имена включают *название поставщика инфраструктуры, ресурс и инструмент* или *назначение*. На рис. 5.4 имя `gcp-server-module` указывает на GCP как на поставщика инфраструктуры, на сервер как на тип ресурса и на модуль как на назначение.

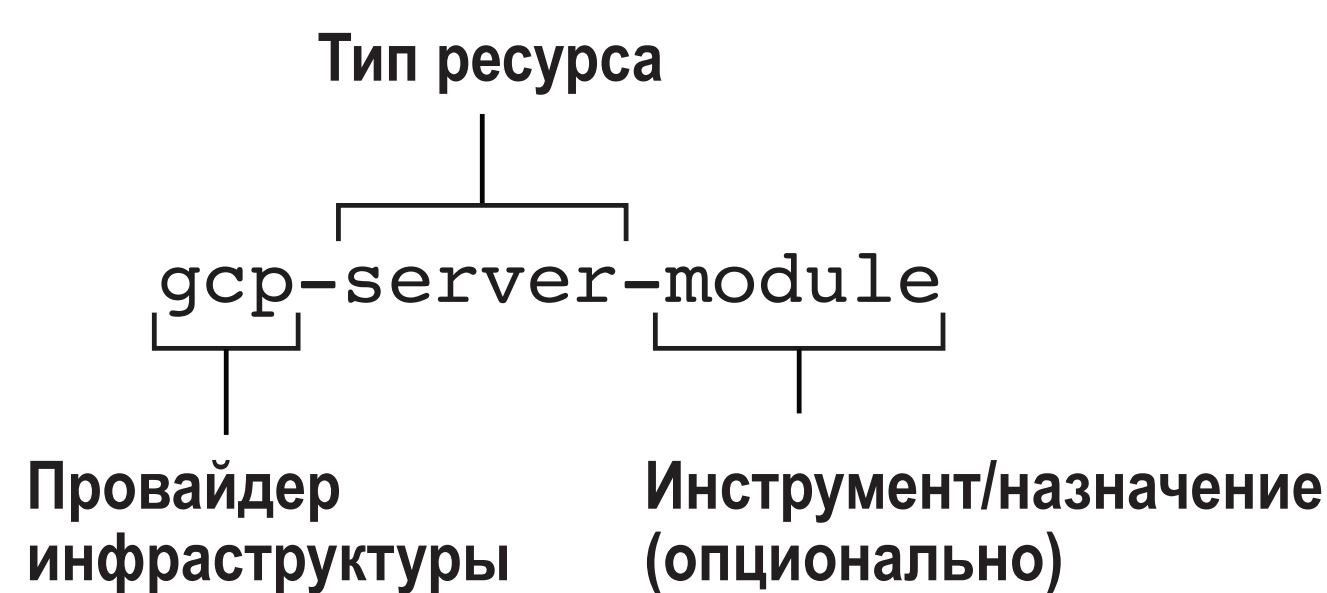


Рис. 5.4 Имя репозитория должно содержать название поставщика инфраструктуры, тип ресурса и назначение

Если ваши модули используют специфический инструмент или имеют уникальное назначение, советую добавить соответствующее обозначение в конец имени репозитория. Добавление названия инструмента к имени помогает определить тип модуля. По аналогии с практиками, описанными в главе 2, модуль должен иметь достаточно информативное имя, чтобы коллеги по команде с первого взгляда понимали его тип и назначение.

Вы можете применить аналогичный подход к именованию папок в одиночном репозитории. Но в данном случае удобнее использовать вложенные каталоги – это облегчает разделение по провайдерам инфраструктуры и типам ресурсов. В зависимости от предпочтений вашей организации и команды вы всегда можете добавить дополнительные поля к имени репозитория.

5.1.3 Выбор структуры репозитория

Масштабируемость вашей системы и фреймворка непрерывной интеграции зависят от того, используете ли вы один или несколько репозитория. Структура Datacenter for Veggies начиналась с единого

репозитория, который работал хорошо, поскольку хранил десятки модулей и несколько сред. Каждый модуль нуждается в средах разработки и промышленных средах. Для каждой среды требуется несколько серверов, одна база данных, сеть и система мониторинга.

Использование единого репозитория дает несколько преимуществ. На рис. 5.5 показаны некоторые преимущества и ограничения. Во-первых, любой член вашей команды может получить доступ к модулям и конфигурациям в едином репозитории. Во-вторых, вам нужно посетить только одно место, чтобы сравнить и выявить различия между средами. Например, вы можете сравнить два файла в репозитории и обнаружить, что в среде разработки всего три сервера, а в промышленной среде – пять.

Основываясь на принципах IaC, структура единого репозитория по-прежнему соответствует принципам компонуемости, развиваемости и воспроизводимости. Любой член команды может зайти в каталог и обновить модуль. Вы спокойно можете создавать один модуль на основе другого, поскольку вся команда использует одинаковое представление инфраструктуры и конфигурации.

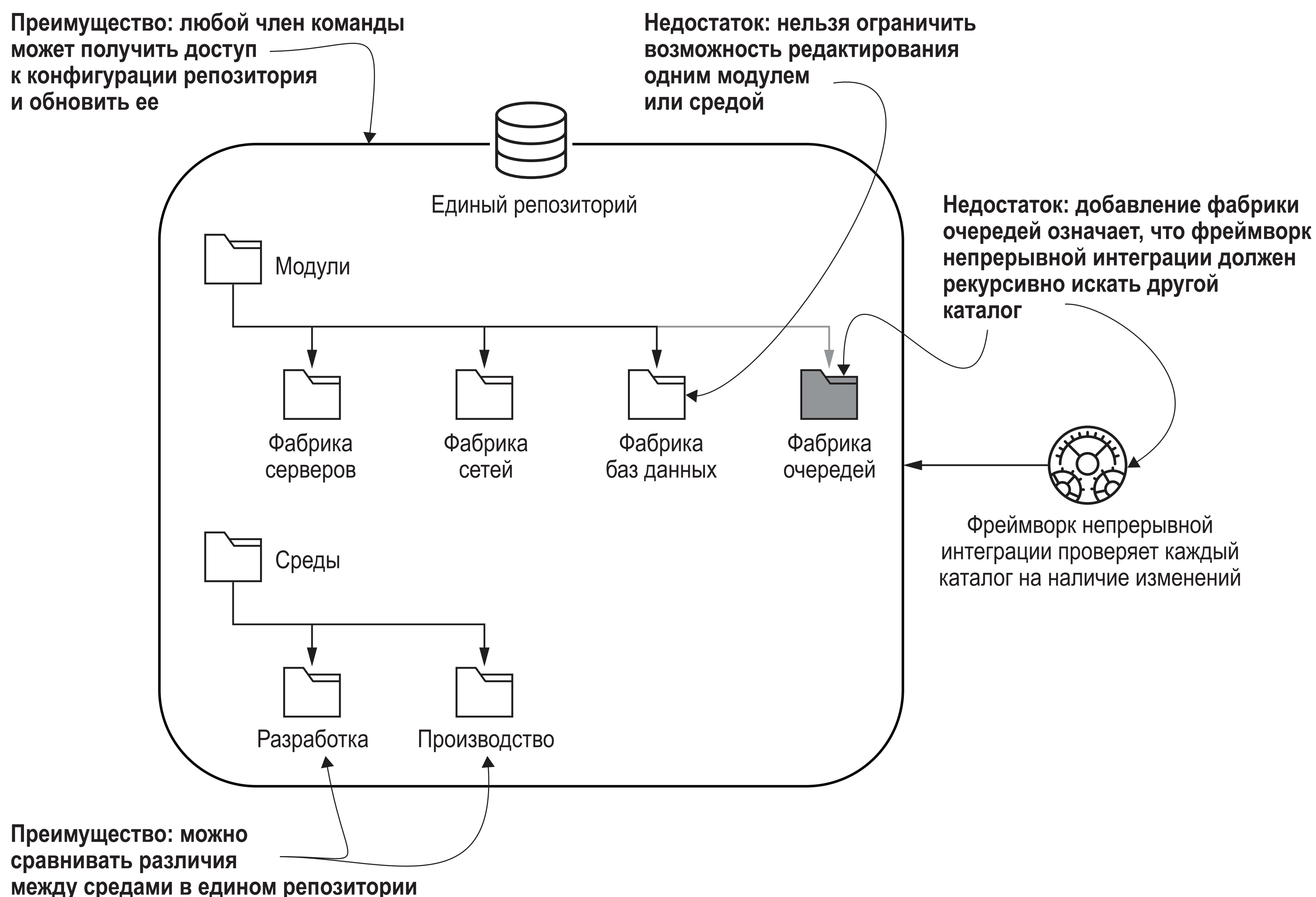


Рис. 5.5 Единый репозиторий обеспечивает одинаковое представление для всех модулей и конфигураций, но ограничивает возможности непрерывной интеграции и детального контроля доступа

С другой стороны, единая структура репозитория имеет некоторые ограничения. Если кто-то неудачно изменит модуль, это мо-

жет сломать сегмент IaC, который от него зависит! Более того, ваша система непрерывной интеграции тоже может выйти из строя, поскольку она рекурсивно проверяет каждый каталог на наличие изменений.

В результате вы можете прийти к решению внедрить методы и инструменты для работы с отдельными репозиториями. К ним относятся строгое управление версиями и специализированные системы сборки. Если вашей организации сложно создать или внедрить инструмент, облегчающий управление одиночным репозиторием, вы всегда можете перейти на структуру с несколькими репозиториями.

ПРИМЕЧАНИЕ Существуют инструменты, облегчающие создание и работу с отдельными репозиториями. У них есть дополнительный функционал для обработки вложенных подкаталогов и отдельных рабочих процессов сборки. К ним относятся, в числе прочих, Bazel, Pants и Yarn.

Миграция на структуру с отдельными репозиториями происходит чаще, чем вы думаете. Мне пришлось сделать это дважды! Одна организация начала с трех сред и четырех модулей. За несколько лет ее IaC разрослась до сотен модулей и сред.

В этих условиях фреймворку CI (Jenkins) требовалось почти три часа, чтобы выполнить стандартное масштабирование серверов. Фреймворк тратил большую часть времени на поиск изменений в каждом каталоге и его подкаталогах! В конечном итоге мы реорганизовали конфигурации и модули в несколько репозиториев.

Рефакторинг с переходом на отдельные репозитории позволил устранить проблемы с CI. Структура с отдельными репозиториями также дала нам возможность более детально управлять доступом к конкретным модулям. Команда безопасности смогла предоставить доступ к редактированию модуля только некоторым командам. Вы узнаете больше о рефакторинге в главе 10.

На рис. 5.6 показаны преимущества и недостатки структуры отдельных репозиториев, включая детальный контроль доступа и масштабирование рабочих процессов CI. В то же время наличие отдельных репозиториев снижает единообразие представлений модулей и конфигураций вашей организации.

Рефакторинг конфигурации в структуру с несколькими репозиториями позволяет изолировать доступ и развивать конфигурацию инфраструктуры для конкретных групп. Вы получаете более строгий контроль над развитием и жизненным циклом модулей. Большинство платформ CI поддерживают несколько репозиториев и запускают рабочие процессы параллельно, когда платформа обнаруживает изменения в данном репозитории.

Однако использование нескольких репозиториев имеет некоторые недостатки. Представьте себе, что Datacenter for Veggies имеет

десять или более модулей в разных репозиториях. Как узнать, все ли они соответствуют одинаковым стандартам формата файлов и правилам именования?

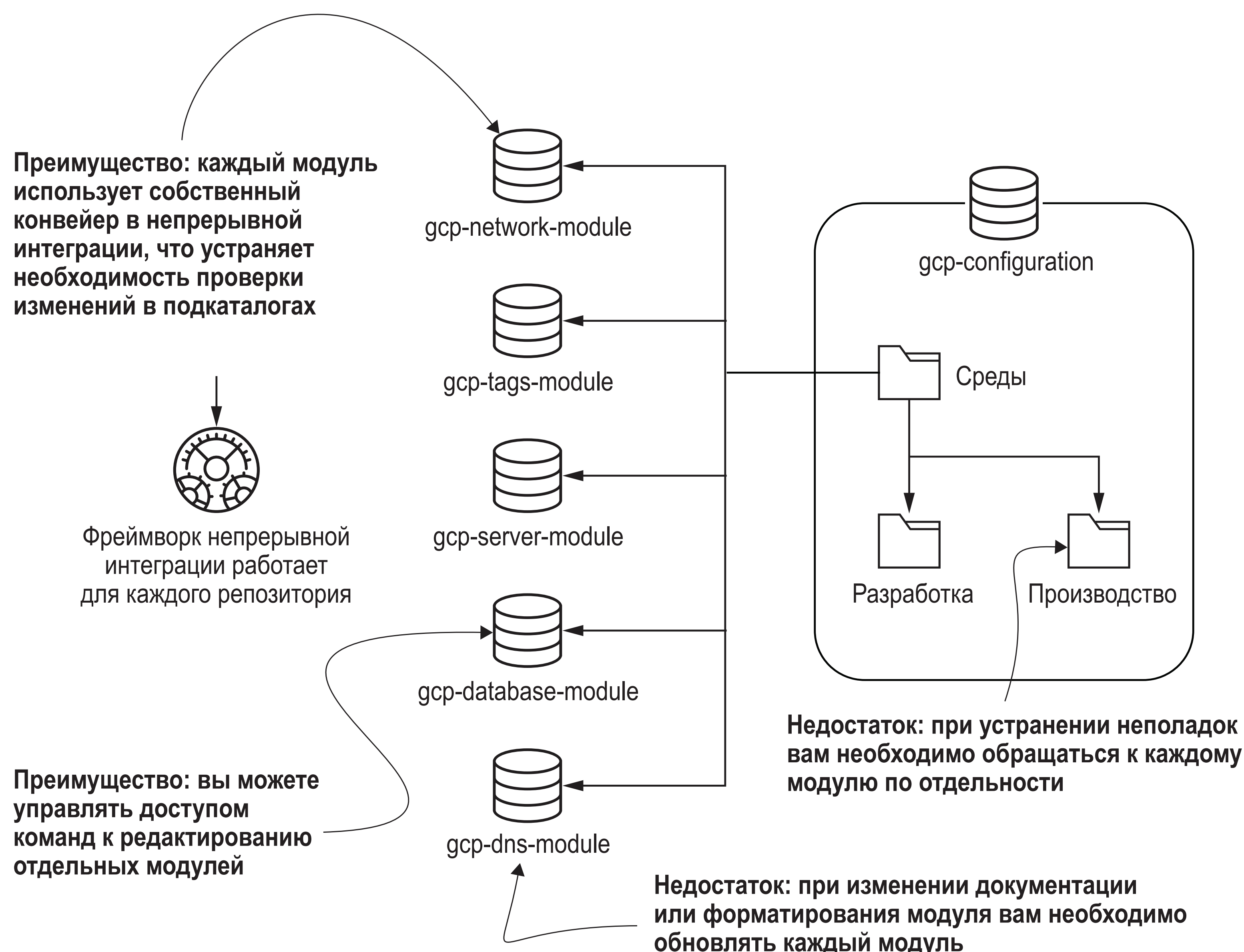


Рис. 5.6 Раздельные репозитории помогают снизить нагрузку на выполнение тестов и конфигураций с помощью CI-фреймворков, но требуют постоянной проверки правильности форматирования и устранения неполадок

На рис. 5.7 показано одно из возможных решений проблемы соответствия файлов общим стандартам. Вы можете записать все тесты форматирования и проверки в модуль-прототип. Затем фреймворк CI загружает тесты и проверяет наличие файлов README и Python в модулях сервера, сети, базы данных и DNS.

Модуль-прототип с тестами помогает проконтролировать формат старых модулей, которые вы используете не так часто, например DNS. Если вы хотите добавить новый стандарт, достаточно дополнить модуль-прототип новым тестом. В следующий раз, когда кто-то будет обновлять модуль или конфигурацию, ему придется обновить также формат модуля, чтобы он соответствовал требованиям.

Стандартизированный набор проверок облегчает операционную нагрузку по поиску и замене файлов в сотнях репозиториях. Он распределяет ответственность за обновление репозитория модуля между всеми участниками разработки модуля. Дополнительная информация о тестировании соответствия и интеграции модулей в ваш рабочий процесс представлена в главах 6, 7 и 8.

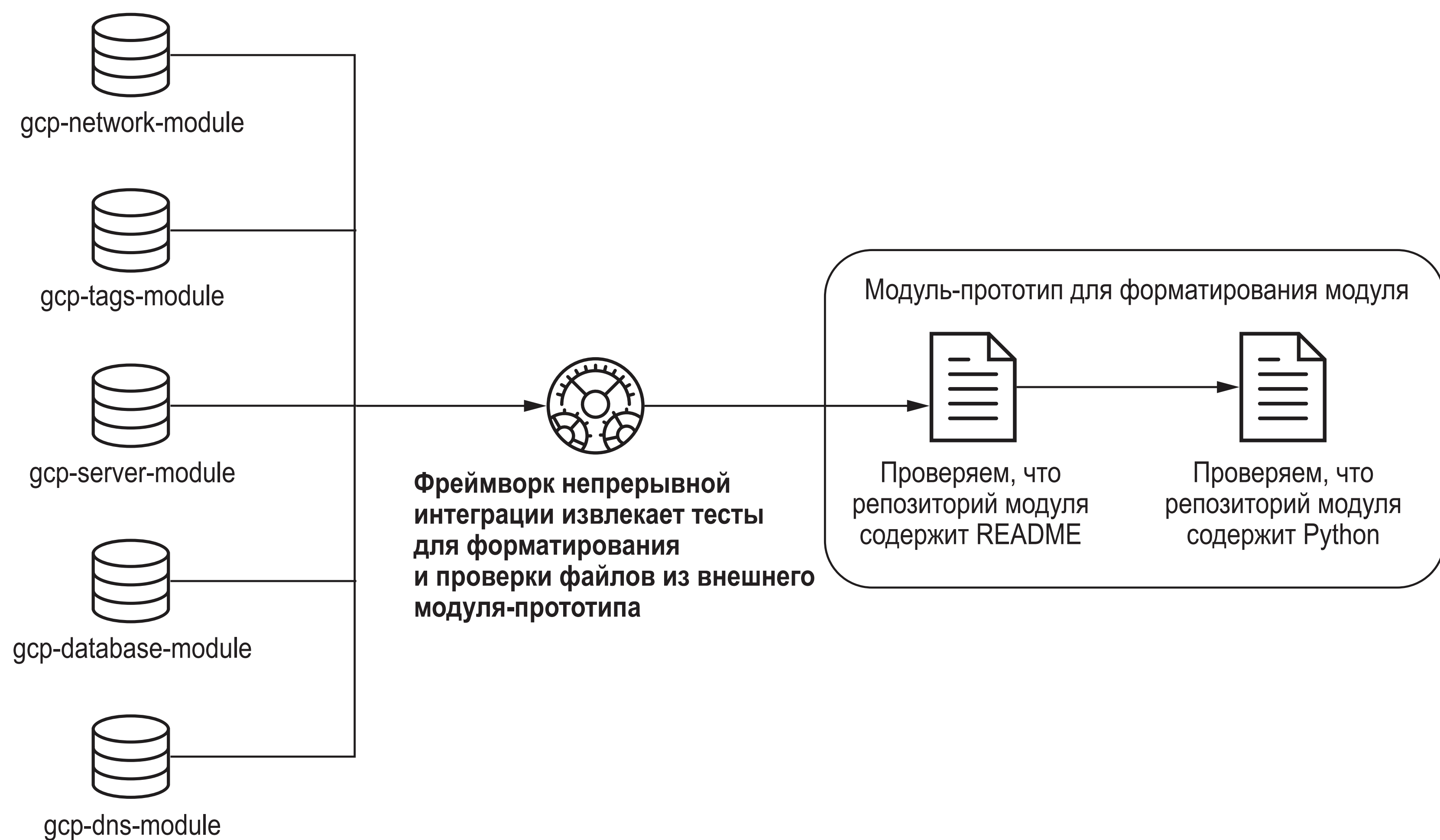


Рис. 5.7 Создание модуля-прототипа, содержащего все проверки формата репозитория модуля, помогает исправить старые репозитории, не соответствующие новым стандартам

Второй недостаток отдельных репозиторий связан с проблемой устранения неполадок. Когда вы ссылаетесь на модуль в своей конфигурации, вам необходимо найти репозиторий модуля, чтобы определить, какие входы и выходы ему необходимы. При устранении ошибок в конфигурации поиск репозиторий отнимает дополнительное время.

Если у вас есть система сборки, которая соответствует вашим требованиям к созданию единого репозитория, вы можете ограничиться одним репозиторием. Однако большинство систем сборки плохо справляются с рекурсивным поиском по каталогам. Чтобы решить эту проблему, вы можете использовать комбинацию одного и нескольких репозиторий.

Допустим, в компании Datacenter for Veggies решили прибегнуть к комбинации репозиторий. Они разделяют каждую конфигурацию для разных типов зелени и фруктов. Команда Leafy Greens использует один репозиторий, а команда Fruits – другой. Обе они применяют общие модули для сети, тегов, базы данных и DNS.

На рис. 5.8 показано, что команде Fruits нужна очередь, а команде Leafy Greens – нет. В результате в репозитории Fruits появился локальный модуль для создания очередей. Команда Fruits использует один репозиторий для своих уникальных конфигураций, но обращается к нескольким репозиториям за общими модулями.

При использовании этого смешанного подхода определите, какой тип управления доступом вам нужен для отдельных репозиторий или общих конфигураций. Если вы хотите улучшить компонентность и воспроизводимость для других команд, имеет смысл поместить модуль в отдельный репозиторий и открыть командам доступ

к нему. Но если вы хотите обеспечить устойчивую развиваемость специфической конфигурации, лучше ограничиться локальным доступом к модулю.

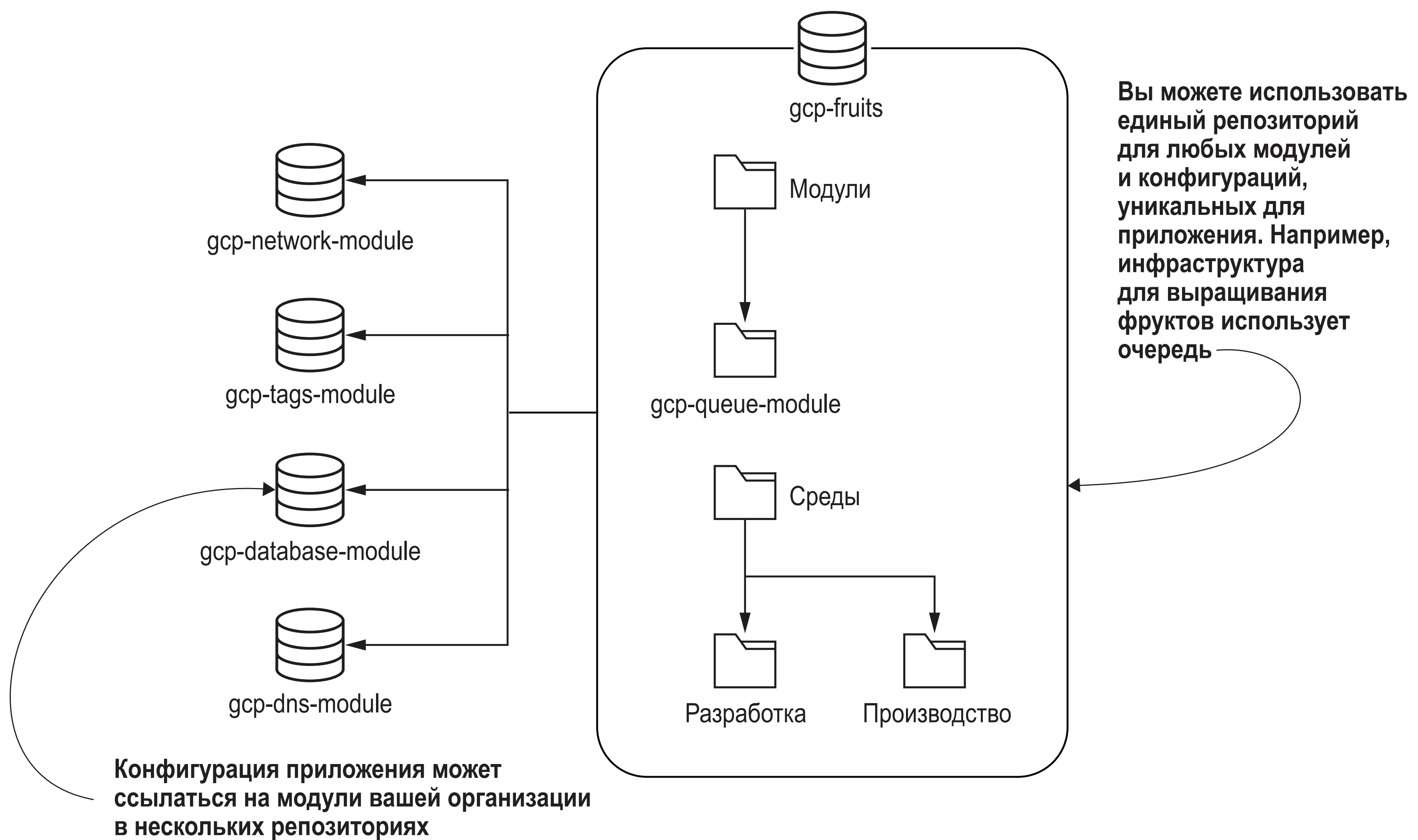


Рис. 5.8 Ваша организация может использовать комбинацию единого и отдельных репозиториев, если это удобно для конфигурации системы или приложения

Выбирая структуру репозитория, учитывайте компромисс между подходами и проводите рефакторинг по мере роста количества модулей и конфигураций. Если вы регулярно добавляете конфигурации и ресурсы в одиночный репозиторий, необходимо убедиться, что инструменты и процессы масштабируются вместе с ним!

5.2 Управление версиями

На протяжении всей этой главы вы использовали практику хранения конфигурации инфраструктуры или кода в системе контроля версий. Например, команды разработчиков Datacenter for Veggies всегда могут найти элемент инфраструктуры с помощью хеша коммита. Однажды команда безопасности Datacenter for Veggies выразила обеспокоенность по поводу чрезмерного времени жизни имен пользователей и паролей к базам данных мониторинга почвы.

Команда безопасности рекомендует использовать менеджер секретов для хранения и смены пароля каждые 30 дней. Проблема заключается в том, что модуль базы данных мониторинга почвы используют все команды. На рис. 5.9 показано, что приложения в на-

стоящее время получают информацию от модуля базы данных. Модуль выводит пароль к базе данных, который приложения используют для записи и чтения данных. Команда безопасности хочет, чтобы вы вместо этого использовали менеджер секретов.

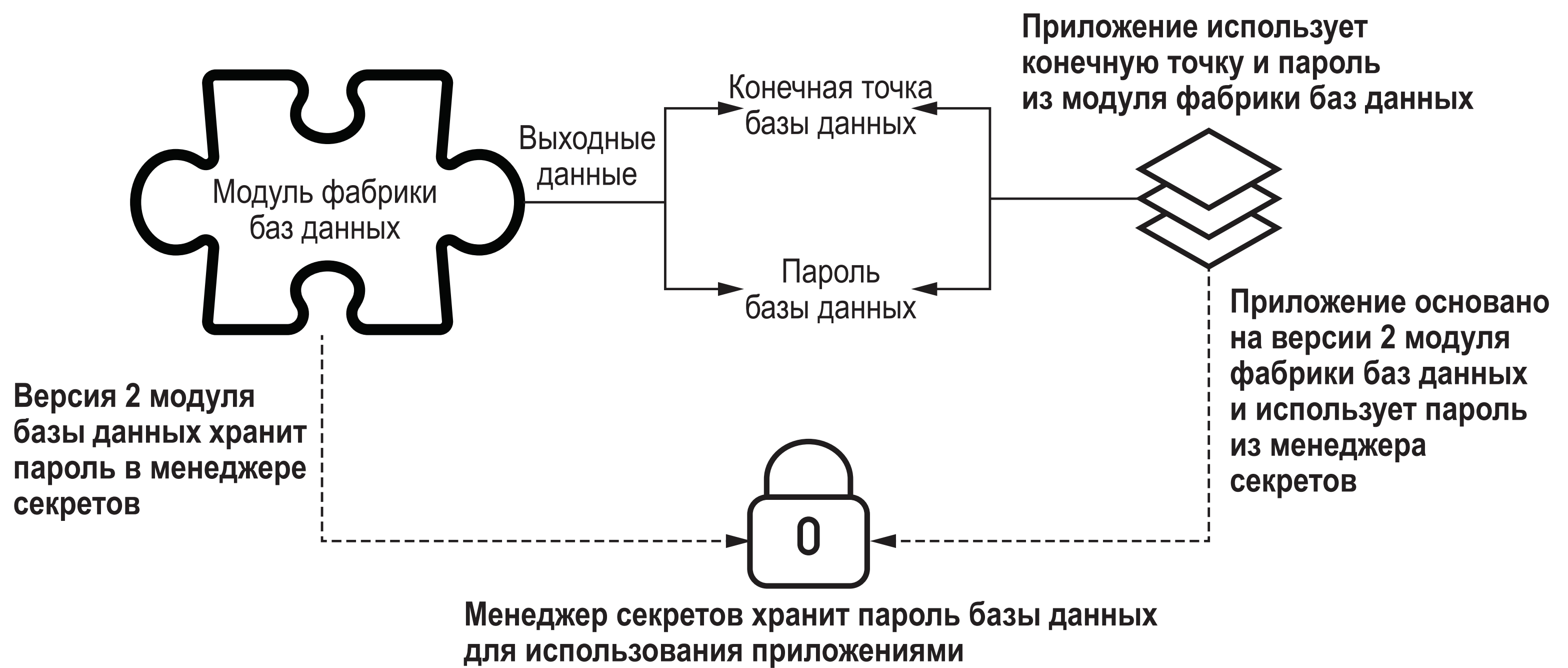


Рис. 5.9 Приложения получают конечную точку базы данных и пароль от модуля мониторинга почвы, но должны использовать пароль, получаемый от менеджера секретов

Выходные данные модуля базы данных влияют на развиваемость и безопасность системы. Как нам обновить базу данных, чтобы использовать менеджер секретов, не нарушая сбор данных о почве? Команда инфраструктуры Datacenter for Veggies решает добавить в модуль базы данных управление версиями.

ОПРЕДЕЛЕНИЕ Управление версиями (versioning) – это процесс назначения уникальных версий итерациям кода.

Давайте рассмотрим, как команда Datacenter for Veggies реализует управление версиями модулей. Команда помечает текущую версию модуля базы данных как v1.0.0. Версия v1.0.0 будет выводить пароль базы данных для приложений:

```
$ git tag v1.0.0
```

Команда отправляет тег v1.0.0 в систему контроля версий:

```
$ git push origin v1.0.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
* [new tag]          v1.0.0 -> v1.0.0
```

Вам необходимо выполнить рефакторинг конфигураций для команд Fruit, Leafy Greens, Grain и Herb, чтобы использовать версию модуля базы данных v1.0.0 в процессе, называемом *фиксацией* (замораживанием, закреплением) *версии* (version pinning). Фиксация версий сохраняет идемпотентность. При запуске IaC конфигурации

продолжают использовать выходные данные модуля базы данных. Вы не должны обнаружить никакого дрейфа между зафиксированным модулем и существующей инфраструктурой.

После того как все команды закрепят у себя v1.0.0, вы можете переписать модуль, чтобы использовать менеджер секретов. Модуль базы данных хранит пароль в менеджере секретов. Команда помечает тегом версии v2.0.0 новый модуль базы данных, который по запросу возвращает конечную точку базы данных и расположение пароля в диспетчере секретов:

```
$ git tag v2.0.0
```

Команда отправляет тег v2.0.0 в систему контроля версий:

```
$ git push origin v2.0.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
* [new tag]          v2.0.0 -> v2.0.0
```

Вы можете изучить разницу между двумя версиями модуля, глядя на историю коммитов:

```
$ git log --oneline
7157d3e (HEAD -> main, tag: v2.0.0, origin/main)
➡Change database module to store password in secrets manager
5c5fd65 (tag: v1.0.0) Add database factory module
```

После создания новой версии модуля фабрики базы данных вы просите несколько команд опробовать ее. Команда Fruits смело становится добровольцем. В настоящее время команда Fruits использует версию 1.0.0. Эта версия модуля возвращает конечную точку базы данных и пароль. При обновлении модуля до версии 2.0.0, как показано на рис. 5.10, команде Fruits необходимо учитывать изменения в рабочем процессе модуля. Команда больше не может напрямую использовать пароль в выходных данных модуля. Модуль возвращает путь API к паролю базы данных, хранящемуся в менеджере секретов. Поэтому команда Fruits обновляет IaC, чтобы перед созданием базы данных получать ее пароль от менеджера секретов.

Вы должны запомнить несколько важных приемов управления версиями модулей. Во-первых, перед обновлением запустите IaC и устраните любой дрейф. Во-вторых, применяйте такой подход к управлению версиями, при котором вы не обращаетесь к последней версии модуля.

Datacenter for Veggies использует *семантическое управление версиями* (semantic versioning), присваивая номера версий, которые передают важную информацию о конфигурации. Вы можете указать версии модуля несколькими способами, в том числе пометить коммит номером в системе контроля версий или упаковать и пометить модуль в репозитории артефактов.

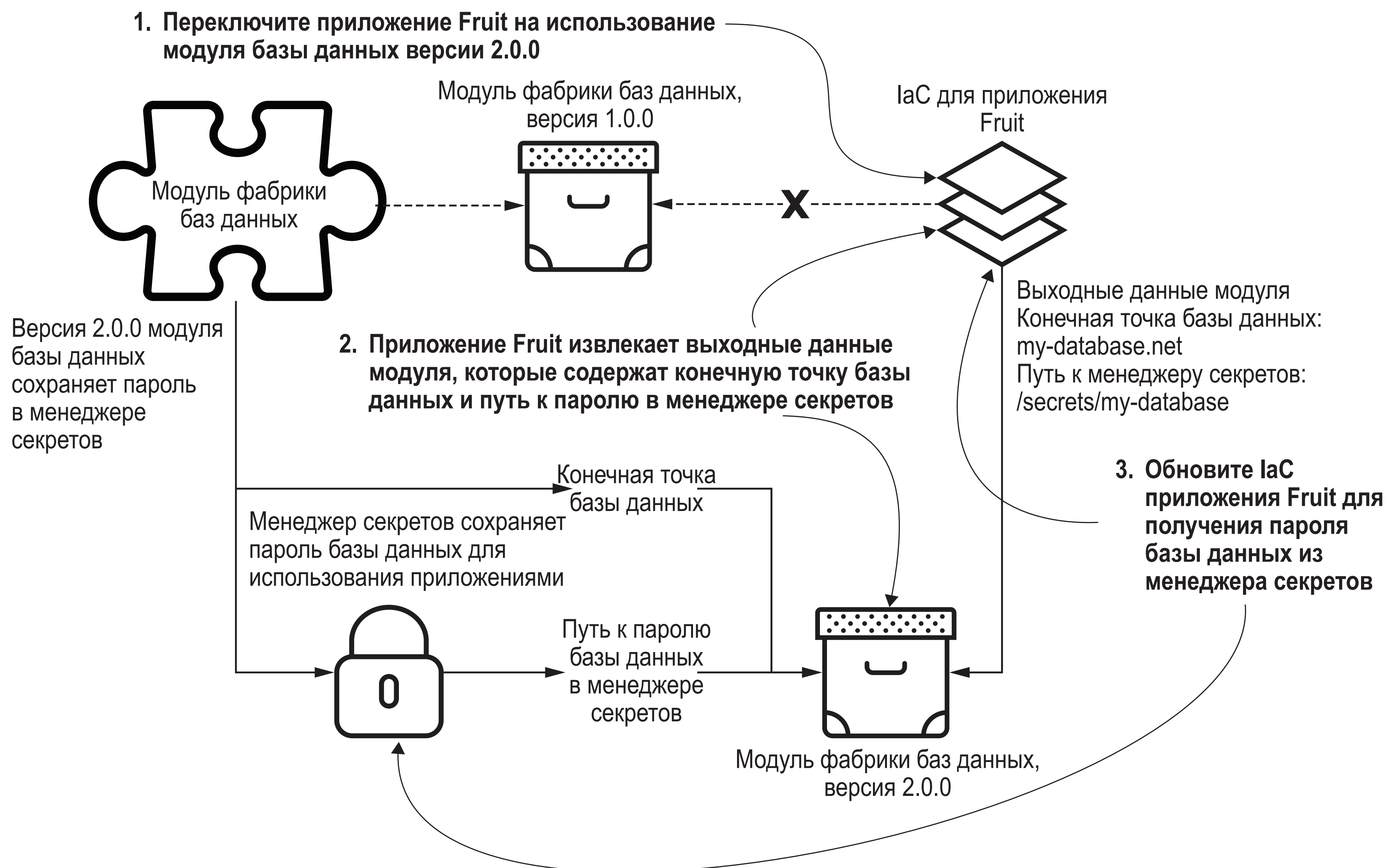


Рис. 5.10 Вы можете провести рефакторинг приложения Fruit, чтобы использовать версию 2.0.0 модуля базы данных и получать пароль базы данных от менеджера секретов

ПРИМЕЧАНИЕ Обычно я выпускаю старшую версию при значительных изменениях, которые удаляют входные, выходные данные и ресурсы, и второстепенную версию, если обновляю значения конфигурации, входные или выходные данные для модулей, которые не влияют на зависимости, используя предыдущие версии. Наконец, я выпускаю патч при незначительных изменениях конфигурации, относящейся к модулю и его ресурсам. Дополнительные сведения о семантическом управлении версиями можно найти по адресу <https://semver.org/>.

Используя продуманный подход к управлению версиями, вы можете более эффективно развивать нижестоящие ресурсы инфраструктуры, не нарушая работу вышестоящих ресурсов, поскольку вы контролируете их зависимости. Управление версиями также помогает при аудите активных версий. Механизм управления версиями позволяет выявлять и объявлять устаревшими старые, неактивные версии модуля. Это помогает сэкономить ресурсы, уменьшить путаницу и распространять последние обновления.

Однако вы должны иметь в виду некоторые особенности управления версиями. Чем дольше вы откладываете обновление приложения для использования базы данных версии 2.0.0, тем выше вероятность сбоя после обновления. Вы можете заранее спланировать,

как долго будет разрешено использовать версию модуля v1.0.0 после появления следующей версии. Вам не нужно сразу удалять версию v1.0.0 модуля базы данных. Обычно я обновляю зависимые модули с небольшими изменениями версий. Чем больше разрыв между версиями, тем больше риск сбоя и доля неудачных изменений.

ПРИМЕЧАНИЕ Если вы практикуете разработку на основе целевых характеристик или Git Flow, вы можете размещать патчи и «заплаты» точно так же, как и при разработке программного обеспечения. Вы можете создать ветку на основе тега версии, внести изменения, увеличить версию исправления и добавить новый тег для ветки патча. Вам нужно будет сохранить ветку для истории коммитов.

Этот процесс управления версиями хорошо работает для структуры с отдельными репозиториями. А как насчет единого репозитория? Вы по-прежнему можете применить подход с использованием тегов контроля версий. Возможно, вы решите добавлять префикс к тегу с именем модуля (`module-name-v2.0.0`). Затем вы можете выпустить пакет и выложить свой модуль в репозиторий артефактов. Ваша система сборки упаковывает содержимое подкаталога модуля и помечает версию в хранилище артефактов. Ваша конфигурация ссылается на удаленные модули в репозитории артефактов, а не на локальный файл.

5.3 Релиз

Я уделила столько внимания управлению версиями модулей, чтобы помочь в их развитии и свести к минимуму сбои в работе вашей системы. Но вам не нужно, чтобы каждая команда немедленно обновляла свой IaC до новейшего модуля. Вместо этого вам достаточно убедиться, что модуль работает и не нарушает вашу инфраструктуру, прежде чем использовать его в промышленной среде.

На рис. 5.11 показано, что вы проверяете обновление модуля базы данных, прежде чем разрешить всем командам Datacenter for Veggies использовать его. После обновления модуля базы данных для хранения пароля в диспетчере секретов вы передаете изменения в систему контроля версий. Вы просите команду Fruits протестировать модуль в отдельной среде и убедиться, что модуль работает. Они подтверждают, что все работает правильно. Вы помечаете выпуск новой версией 2.0.0 и обновляете документацию по менеджеру секретов.

К настоящему моменту команда инфраструктуры Datacenter for Veggies обновила модуль и сначала протестировала его в среде разработки команды Fruits. Теперь, когда модуль прошел тестирова-

ние, другие команды могут использовать новый модуль базы данных с менеджером секретов. Команда осуществила процесс *выпуска*, чтобы убедиться, что другие команды могут использовать новый модуль.

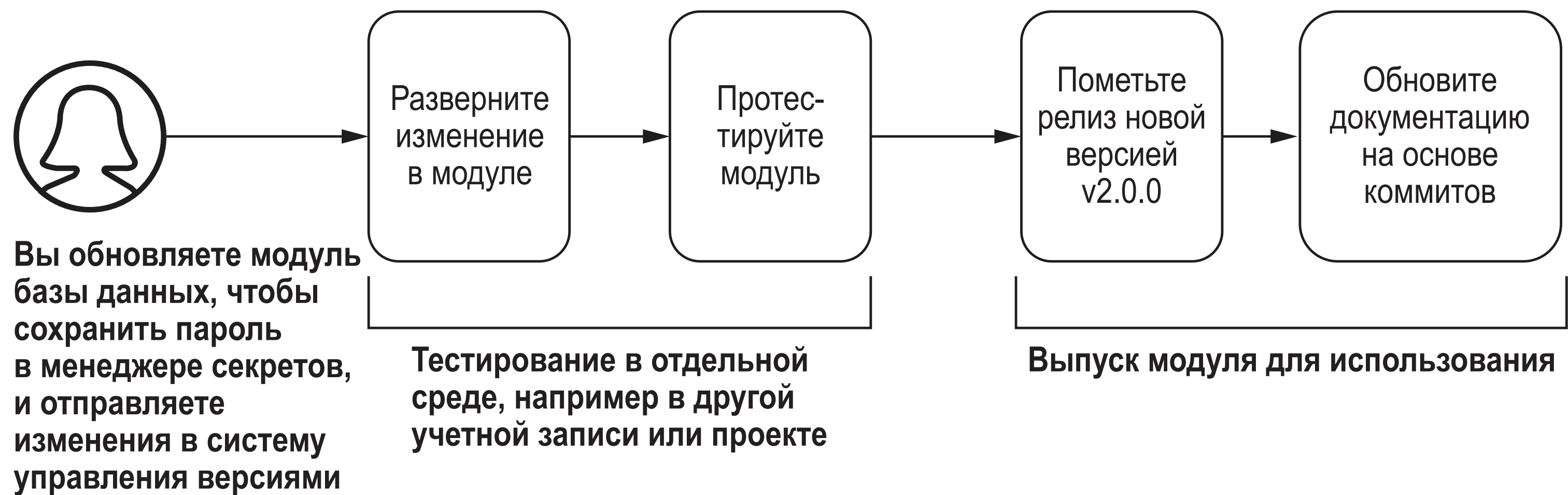


Рис. 5.11 При обновлении модуля обязательно включите этап тестирования перед выпуском модуля и обновлением его документации

ОПРЕДЕЛЕНИЕ *Выпуск* (releasing) – это процесс распространения программного обеспечения среди потребителей.

Процесс выпуска выявляет и изолирует любые проблемы, связанные с обновлениями модулей. Вы не выпускаете пакет с новым модулем, пока тесты не подтвердят его работоспособность.

Я рекомендую запускать тесты модулей в специальной среде тестирования, отдельно от рабочих нагрузок разработки и промышленной инфраструктуры. Отдельная учетная запись или проект для тестирования модулей поможет вам отслеживать стоимость выполнения тестов и изолировать сбои от активных сред. О процессе и средах тестирования более подробно говорится в главе 6.

ПРИМЕЧАНИЕ Подробный листинг кода конвейера непрерывной доставки для выпуска модулей можно найти по адресу <http://mng.bz/PnaR>. Конвейер GitHub Actions автоматически создает релиз GitHub при успешном завершении тестов на основе сообщения о коммите.

После тестирования модулей вы присваиваете выпуску номер версии, который будет использовать ваша команда. Datacenter for Veggies выпускает модуль базы данных версии 2.0.0 и использует диспетчер пакетов Python для привязки тега. В качестве альтернативы вы можете упаковать модуль и отправить его в репозиторий артефактов или в бакет хранилища.

Некоторые организации предпочитают упаковывать артефакт и хранить его в отдельном репозитории для дополнительного контроля безопасности. Если вы пользуетесь защищенной сетью, в которой нет доступа к внешней конечной точке контроля версий, вы

можете вместо этого обратиться к репозиторию артефактов. Просто не забудьте сохранить тег в системе контроля версий, чтобы кто-нибудь мог сопоставить артефакт с правильной версией кода.

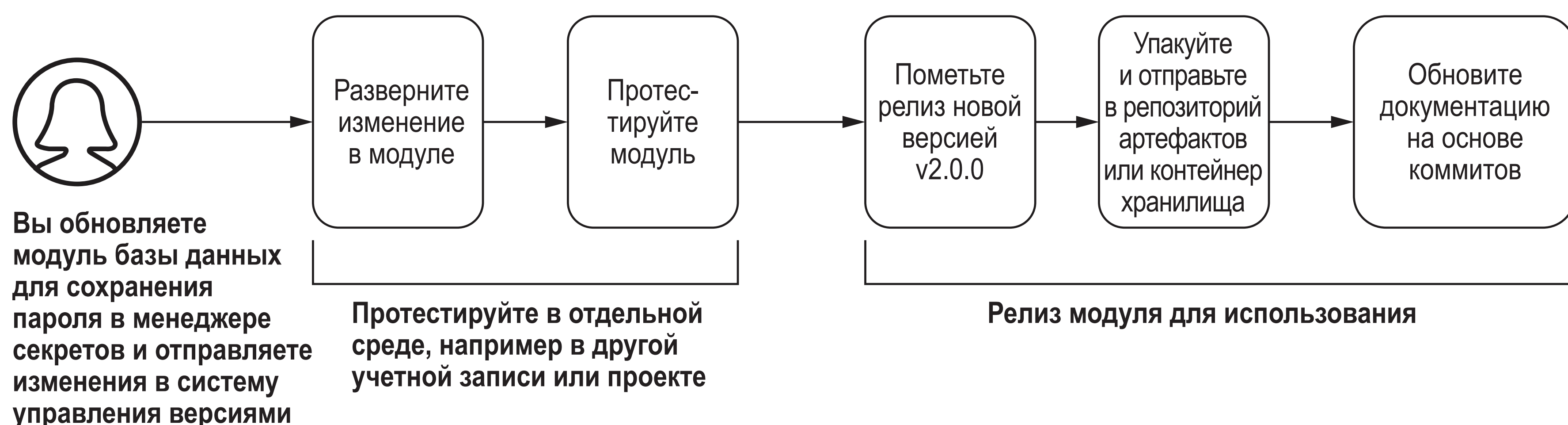


Рис. 5.12 После тестирования вы при желании можете упаковать и отправить модуль в репозиторий артефактов или в бакет хранилища

После упаковки и отправки артефакта вам следует обновить документацию с описанием ваших изменений. Эта документация, называемая примечаниями к выпуску, описывает критические изменения выходных и входных данных. Примечания к выпуску сообщают другим командам сводку изменений.

ОПРЕДЕЛЕНИЕ В *примечаниях к выпуску* (release notes) перечислены изменения в коде для данного выпуска. Вам следует хранить их в репозитории в виде документа, часто называемого *журналом изменений* (changelog).

Вы можете обновлять примечания к выпуску вручную, но я предпочитаю использовать автоматический инструмент семантического выпуска (например, semantic-release), который исследует историю коммитов и создает для меня примечания к выпуску. Важно использовать правильный формат сообщения коммита, чтобы инструмент мог сопоставлять и анализировать изменения. В главе 2 подчеркивалась важность написания содержательных сообщений коммита. Они также пригодятся для инструмента автоматического выпуска.

Допустим, модуль базы данных хранит пароль в менеджере секретов. Datacenter for Veggies считает это важной функцией, поэтому вы добавляете к сообщению коммита префикс feat:

```
$ git log -n 1 --oneline
1b65555 (HEAD -> main, tag: v0.0.1, origin/main, origin/HEAD)
➡ feat(security): store password in secrets manager
```

Анализатор коммита в инструменте автоматического релиза обновляет основную версию тега до версии 2.0.0 на основе этого коммита.

Построение образа

Вы можете столкнуться с практикой использования строителей образов для создания неизменяемых образов сервера или контейнера. Записав нужные пакеты в образ сервера или контейнера, вы можете создавать новые серверы с обновлениями без проблем, связанных с обновлениями на месте. При выпуске неизменяемых образов выполните рабочий процесс создания тестового сервера на основе образа, убедитесь, что он работает правильно, и обновите тег версии образа. Глава 7 описывает некоторые из этих рабочих процессов.

Помимо обновления примечаний к выпуску, обязательно обновляйте сопроводительные файлы и документацию. Сопроводительные файлы помогут вашим товарищам по команде использовать модуль. Например, в Datacenter for Veggies принято, чтобы команды всегда включали в пакет файл README. Этот файл документирует назначение, входные и выходные данные каждого модуля.

ОПРЕДЕЛЕНИЕ *Файл README* – это документ в репозитории, в котором даны инструкции по использованию и добавлению кода. В случае IaC используйте его для документирования назначения, входных и выходных данных модуля.

Для проверки наличия файла README можно использовать правила линтинга. В главе 2 обсуждались некоторые методы проверки, позволяющие обеспечить чистоту IaC. Применение паттерна к сопроводительным файлам и документации помогает форматировать и упорядочивать IaC большого объема.

В примерах Python в состав модулей входят дополнительные файлы, такие как `__init__.py` для идентификации пакета и `setup.py` для конфигурации модуля. Я часто называю файлы с конфигурацией или метаданными, которые помогают конкретным инструментам либо языкам, вспомогательными файлами. Они меняются в зависимости от используемого вами инструмента и платформы. Вам придется их стандартизировать, чтобы средства автоматизации могли их находить и изменять.

5.4 Совместное использование модулей

По мере того как Datacenter for Veggies выращивает больше продукции, компания нанимает *новые команды*, которые автоматизируют выращивание зерновых, чая, кофе и бобов. Компания также создает новую команду для исследования продукции из диких сортов растений. Каждая команда должна иметь возможность расширять существующие модули, а также создавать новые.

Например, команде Beans (зерновые) необходимо изменить модуль базы данных для использования PostgreSQL версии 12. Должны ли члены этой команды иметь возможность редактировать модуль с обновлением версии? Или им следует подать заявку на обновление вам – команде инфраструктуры?

Вам необходимо предоставить различным командам возможность создавать и обновлять модули с помощью IaC. Однако вы должны быть уверены, что команды не изменят важный атрибут и не поставят под угрозу безопасность или функциональность. Далее мы рассмотрим несколько приемов, которые помогут вам наладить совместное использование модулей в вашей организации.

Представьте, что всем командам Datacenter for Veggies нужна база данных. Вы создаете новый модуль базы данных, который задает набор параметров по умолчанию для обеспечения безопасности и функциональности. Модуль базы данных использует встроенные значения по умолчанию для входных данных модуля, чтобы охватить множество вариантов использования Datacenter for Veggies. Даже если команда Coffee не знает, как создать базу данных, она может использовать готовый модуль для создания безопасной и работающей базы данных.

Как правило, в модуле указывают несколько чрезмерные настройки параметров безопасности по умолчанию, потому что всем хочется перестраховаться. Если команде требуется больше гибкости, она может обновить модуль или переопределить атрибуты по умолчанию. Предустановленные значения по умолчанию помогают обучить команды безопасным и стандартным методам развертывания определенных ресурсов инфраструктуры.

В нашем сценарии команда Beans заявляет о необходимости большей гибкости. Модуль не использует новую версию базы данных PostgreSQL версии 12. Ни одна другая команда не использует эту версию PostgreSQL. Команда Beans решает обновить версию базы данных и отправить изменения в репозиторий.

Однако изменения не вступают в силу немедленно. Компоновщик отправляет уведомление ответственному за утверждение изменений в группе инфраструктуры. На рис. 5.13 показано, что команда инфраструктуры приостанавливает компоновщик и просматривает изменения. Если изменения получают одобрение команды, компоновщик выпускает модуль. Теперь команда Beans может использовать новую версию модуля базы данных с PostgreSQL версии 12.

Почему вы должны разрешить команде Beans изменять модуль инфраструктуры? Самообслуживание изменений модулей дает всем командам возможность обновлять свои системы и снижает нагрузку на команды, занимающиеся инфраструктурой и платформой. Вам необходимо найти разумный баланс между удобством и скоростью разработки, с одной стороны, и безопасностью инфраструктуры – с другой. Добавление шага проверки и согласования перед выпуском

модуля позволяет выявить потенциальные сбои или нестандартные изменения в инфраструктуре.

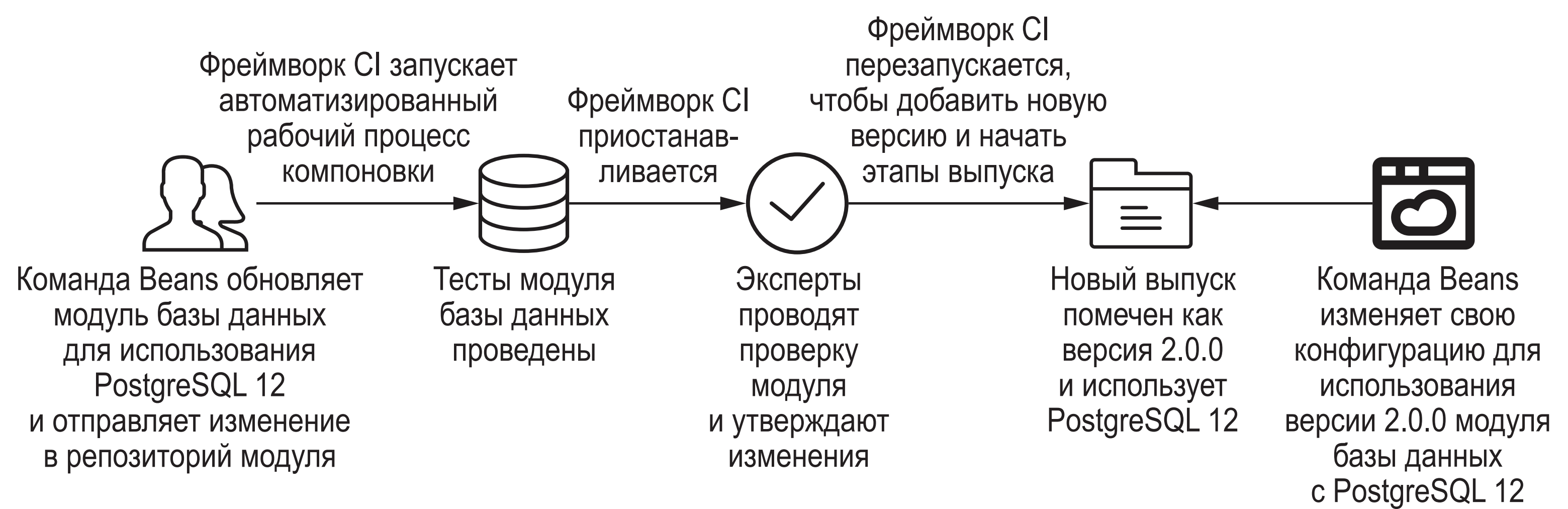


Рис. 5.13 Команда приложения может обновить модуль базы данных. Однако команде придется дождаться одобрения профильных экспертов, прежде чем она сможет использовать новую версию

Подход, при котором любой команде разрешено использовать модули и редактировать их под надзором проверяющих, лучше всего работает при наличии четких стандартов и процессов разработки модулей. Если вы не установите строгие стандарты разработки модулей, этот подход постепенно приведет к путанице и усложнит реализацию изменений инфраструктуры в промышленной среде.

Вернемся к нашему примеру. Группа инфраструктуры не очень доверяет этим изменениям, поэтому просит администратора базы данных провести дополнительную проверку. Администратор базы данных указывает, что если команда Beans обновит версию своего модуля, в результате будет удалена предыдущая база данных и создана пустая база данных с новой версией! Это существенно нарушит работу приложения, поддерживающего выращивание зерновых.

На рис. 5.14 команда Beans отправляет запрос на помощь команде базы данных. Администратор рекомендует методы, позволяющие обновить базу данных без удаления данных. Команда Beans реализует эти методы и просит провести повторную проверку. Как только модуль будет выпущен, команда сможет использовать его, не беспокоясь о нарушении работы своих приложений.

Если у вас есть опасения, что изменение может особенно сильно повлиять на архитектуру, безопасность или доступность системы, запросите проверку у профильного эксперта, прежде чем выпускать новую версию. Эксперт может выявить проблемы, способные повлиять на другие команды, использующие модуль, и предложить наиболее безопасный способ обновления. Процесс проверки помогает развивать IaC и выявлять потенциальные сбои, связанные с изменениями инфраструктуры.

В общем, вам нужен процесс, который позволит вашей команде вносить изменения в инфраструктуру и предоставит ей знания и поддержку для успешного завершения этих изменений без нарушения работы критически важных систем. Ручная проверка может

показаться утомительной, но она помогает обучить вашу команду и предотвратить проблемы в производстве. Ваша команда должна найти баланс между быстрым развертыванием изменений в рабочей среде и ожиданием ручной проверки от эксперта, о чем я подробнее расскажу в главе 7.

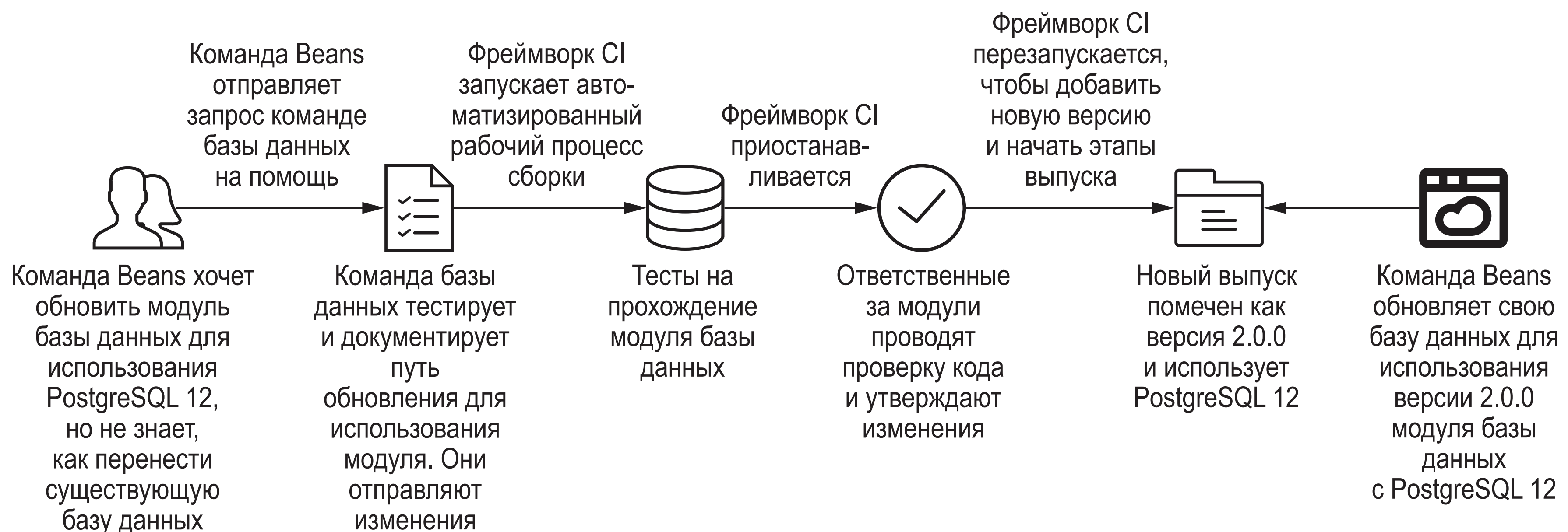


Рис. 5.14 В случае потенциально опасных обновлений модуля команда приложения отправляет запрос группе базы данных для проверки шагов миграции базы данных перед выпуском новой версии модуля

Совместная работа над модулями заставляет команды обмениваться знаниями и учит их коллективно выявлять потенциальные сбои в критически важной инфраструктуре. Вы можете рассматривать модули как артефакты для использования в масштабе организации, подобно общим библиотекам приложений, образам контейнеров или образам виртуальных машин. Любой сотрудник компании может использовать и обновлять модули (при необходимости с дополнительной помощью!), если это необходимо для развития архитектуры инфраструктуры, безопасности или доступности.

Краткое содержание главы

- Совместно используйте модули и конфигурации, размещенные в одном или нескольких репозиториях.
- Структура с единым репозиторием объединяет все конфигурации и модули в одном месте, что упрощает поиск и устранение неисправностей и идентификацию полезных ресурсов.
- Структура с отдельными репозиториями распределяет все конфигурации и модули по разным репозиториям кода в соответствии с бизнес-задачей, функцией, командой или средой.
- Структура с отдельными репозиториями обеспечивает лучший контроль доступа к отдельным конфигурациям инфраструктуры или модулям и оптимизирует выполнение конвейера для каждого репозитория.

- Структура с единым репозиторием может не справиться с масштабированием, когда над IaC начнет работать больше людей, и компоновщику потребуются дополнительные ресурсы для быстрой обработки изменений.
- Реорганизуите один репозиторий в несколько репозиториев, по одному для каждого модуля.
- Выберите единую методологию управления версиями модулей и обновляйте их с помощью тегов Git.
- Упакуйте и выпустите модуль в репозиторий артефактов, что позволит любому сотруднику организации получить нужную версию модуля.
- При совместном использовании модулей разными командами установите в них строгие параметры по умолчанию для обеспечения безопасности и функциональности.
- Разрешите любому сотруднику организации предлагать обновления модулей, но добавьте проверку для выявления потенциально опасных изменений в модулях, которые влияют на архитектуру, безопасность или доступность инфраструктуры.

Тестирование

Основные темы главы:

- варианты тестов, которые следует применять к инфраструктурным системам;
- написание тестов для проверки конфигурации инфраструктуры или модулей;
- стоимость различных типов тестов для компании.

В главе 1 было сказано, что технология IaC представляет собой процесс внесения изменений в систему. Вы обновляете сценарии или конфигурации, передаете их в систему контроля версий, и ваши обновления автоматически применяются к инфраструктуре. Но даже при самом тщательном использовании модулей и паттернов зависимостей из глав 3 и 4 вы можете внести неудачные изменения! Как обнаружить неудачное обновление, прежде чем оно попадет в промышленную среду?

Эту проблему можно решить с помощью тестирования IaC. *Тестирование* – это процесс, который оценивает, работает ли система так, как ожидалось. В этой главе мы рассмотрим различные вопросы тестирования IaC и поставим перед собой задачу снизить частоту неудачных изменений и повысить уверенность в изменениях инфраструктуры.

ОПРЕДЕЛЕНИЕ Тестирование IaC – это процесс проверки, работает ли инфраструктура должным образом.

Представьте, что вы настраиваете сетевой коммутатор с новым сегментом сети. Вы можете вручную протестировать существующие сети, проверив связь с каждым сервером в каждой сети и проверив их подключение. Чтобы проверить правильность настройки новой сети, вы создаете новый сервер и проверяете, отвечает ли он при подключении к нему. Этот ручной тест занимает несколько часов для двух или трех сетей.

По мере создания новых сетей проверка сетевого подключения может занять несколько дней. Для каждого обновления сегмента сети вы должны вручную проверить сетевое подключение, а также серверы, очереди, базы данных и другие ресурсы, работающие в сети. Вы не можете протестировать их все, поэтому проверяете только несколько ресурсов. К сожалению, такой подход может оставить скрытые ошибки или проблемы, которые появятся только через несколько недель или даже месяцев!

Чтобы уменьшить объем ручного тестирования, вы можете автоматизировать тесты, написав скрипты для каждой команды. Ваш скрипт создает сервер в новой сети, проверяет его подключение и тестирует подключения к существующим сетям. Да, вы потратите некоторое время и усилия на написание тестов, но сэкономите часы ручной проверки, запуская автоматический сценарий для любых последующих изменений конфигурации сети.

На рис. 6.1 показано количество затраченных усилий в часах в зависимости от количества ресурсов инфраструктуры при выполнении ручного и автоматического тестирования. Когда вы запускаете сетевые тесты вручную, вам приходится тратить много времени на тестирование. Зависимость почти линейная: чем больше ресурсов в системе, тем выше трудозатраты. Разумеется, написание автоматических тестов тоже требует первоначальных усилий. Но трудозатраты на выполнение теста обычно уменьшаются по мере роста вашей системы. Вы даже можете запускать автоматические тесты параллельно, чтобы сократить общие затраты времени на тестирование.

Конечно, тестирование не выявляет все проблемы и не предотвращает все потенциальные сбои в вашей системе. Здесь важно другое – скрипт автоматического тестирования строго выполняет весь перечень обязательных проверок каждый раз, когда вы вносите изменения. Если обнаруживается скрытая ошибка, вам достаточно слегка доработать тест, чтобы гарантировать, что ошибка больше не повторится! Поэтому автоматическое тестирование со временем снижает общие эксплуатационные трудозатраты.

Вы можете использовать фреймворк тестирования вашего провайдера инфраструктуры, готовые инструменты или собственные библиотеки тестов, написанные на удобном для вас языке программирования. В листингах примеров кода используется среда тестирования Python под названием `pytest` и `Apache Libcloud` – библиотека

Python для подключения к GCP. В этих примерах ключевое значение имеют принципы и объекты тестирования, а не синтаксис кода. Вы можете перенести представленные здесь приемы тестирования на любой другой инструмент или фреймворк.



Рис. 6.1 Ручное тестирование на начальном этапе может потребовать меньших усилий, но по мере увеличения количества инфраструктурных ресурсов в вашей системе эти усилия возрастают. Автоматизированное тестирование требует больших первоначальных усилий, которые уменьшаются по мере роста вашей системы

Подробнее о pytest и Apache Libcloud

Вы найдете инструкции, примеры и зависимости в репозитории примеров кода по адресу <https://github.com/joatmon08/manning-book>. Там есть ссылки для начала работы с pytest и Libcloud.

Не пишите тесты для каждого элемента IaC в вашей системе. Слишком подробные тесты быстро становятся избыточными и трудными в сопровождении. В этой главе я объясню, как определить, когда нужно писать тест и какой тип теста применим к ресурсу, который вы меняете. Тестирование инфраструктуры – это эвристика; вы никогда не сможете полностью предсказать или смоделировать изменения в промышленной среде. Хороший тест дает понимание и практику настройки инфраструктуры или того, как изменение повлияет на систему. Я также отделил тесты, применимые к таким модулям, как фабрики, прототипы или построители, от общих тестов составной или одинарной конфигурации реальной среды.

6.1 Цикл тестирования инфраструктуры

Тестирование помогает обрести уверенность и оценить влияние изменений на инфраструктурные системы. Но разве можно протестировать систему, не создавая ее полностью? Кроме того, как узнать, что система сохранит работоспособность после внесения изменений?

На рис. 6.2 показан цикл тестирования инфраструктуры, предназначенный для структурирования рабочего процесса. После определения конфигурации инфраструктуры вы запускаете начальные тесты для проверки этой конфигурации. Если они пройдут успешно, вы сможете применить изменения к действующей инфраструктуре и протестировать систему.

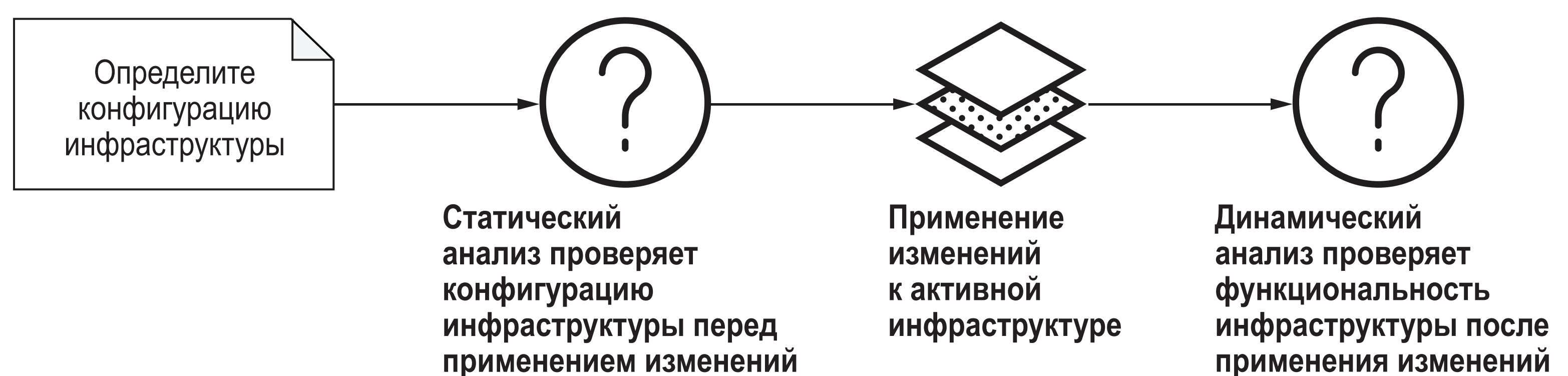


Рис. 6.2 Тестирование инфраструктуры показывает, можем ли мы применить изменения к системе. После применения изменений мы выполняем дополнительные тесты, чтобы подтвердить успешность обновления

В этом рабочем процессе вы выполняете два типа тестов. Один тип статически анализирует конфигурацию перед развертыванием изменений инфраструктуры, а другой динамически анализирует ресурс инфраструктуры, чтобы убедиться, что он по-прежнему работает. Большинство общепринятых тестов следуют этому подходу и выполняются до и после развертывания изменений.

6.1.1 Статический анализ

Как применить цикл тестирования инфраструктуры к нашему примеру с сетью? Представьте, что мы проверяем скрипт настройки сети, чтобы убедиться, что новый сегмент сети имеет правильный диапазон IP-адресов. Нам не нужно развертывать изменения в сети. Вместо этого мы анализируем скрипт, т. е. статический файл.

На рис. 6.3 показано, что мы выбираем сетевой скрипт и запускаем статический анализ. Если обнаружится неправильный IP-адрес, тест будет провален. Мы можем отменить или исправить изменения настроек сети и повторно запустить тесты. Если они пройдут успешно, мы сможем перейти к следующему этапу и применить правильные настройки диапазона IP-адресов к действующей сети.

Исправьте ошибки в конфигурации.
Если их невозможно исправить, отмените
изменения к предыдущей успешной конфигурации



Рис. 6.3 Если статический анализ провален, мы можем либо исправить конфигурацию, чтобы пройти тест, либо вернуться к последней успешной конфигурации

Тесты, которые оценивают конфигурацию инфраструктуры перед развертыванием изменений в ресурсах инфраструктуры, выполняют статический анализ.

ОПРЕДЕЛЕНИЕ *Статический анализ* IaC проверяет конфигурацию инфраструктуры в виде текстового файла перед развертыванием обновлений в действующих ресурсах инфраструктуры.

Статический анализ не требует наличия ресурсов инфраструктуры, поскольку обычно он проверяет конфигурацию. Он не может повлиять на какие-либо действующие системы. Если тесты статического анализа пройдены успешно, это не гарантия будущего успеха, а повод для надежды на безопасное развертывание обновлений.

Я часто использую тесты статического анализа для проверки соблюдения стандартов именования инфраструктуры и зависимостей. Они запускаются до применения изменений и за считанные секунды выявляют любые проблемы с несогласованным именованием или конфигурацией. Я могу исправить ошибки, повторно запустить тесты и, после успешного прохождения, применить изменения к ресурсам инфраструктуры. Вернитесь к главе 2, чтобы освежить в памяти правила чистого кода IaC, проверки и форматирования.

Тесты статического анализа не применяют изменения к активной инфраструктуре, что упрощает откат. Если тесты статического анализа не пройдены, вы можете вернуться к конфигурации инфраструктуры, исправить проблемы и сделать новый коммит. Если не получается исправить конфигурацию для прохождения статического анализа, вы легко вернетесь к предыдущему успешному коммиту! Подробнее об отмене изменений вы узнаете в главе 11.

6.1.2 Динамический анализ

Если статический анализ пройден, вы можете развернуть изменения в сети. Однако вы не знаете, действительно ли сегмент сети ра-

ботает. В конце концов, серверу необходимо подключиться к сети. Чтобы проверить подключение, вы создаете сервер в сети и запускаете тестовый сценарий для проверки входящего и исходящего подключений.

На рис. 6.4 показан цикл тестирования работоспособности сети. После внесения изменений в среду действующей инфраструктуры вы запускаете тесты для проверки функциональности системы. Если тестовый сценарий завершается неудачно и показывает, что сервер не может подключиться, вы возвращаетесь к настройке и исправляете ее для системы.

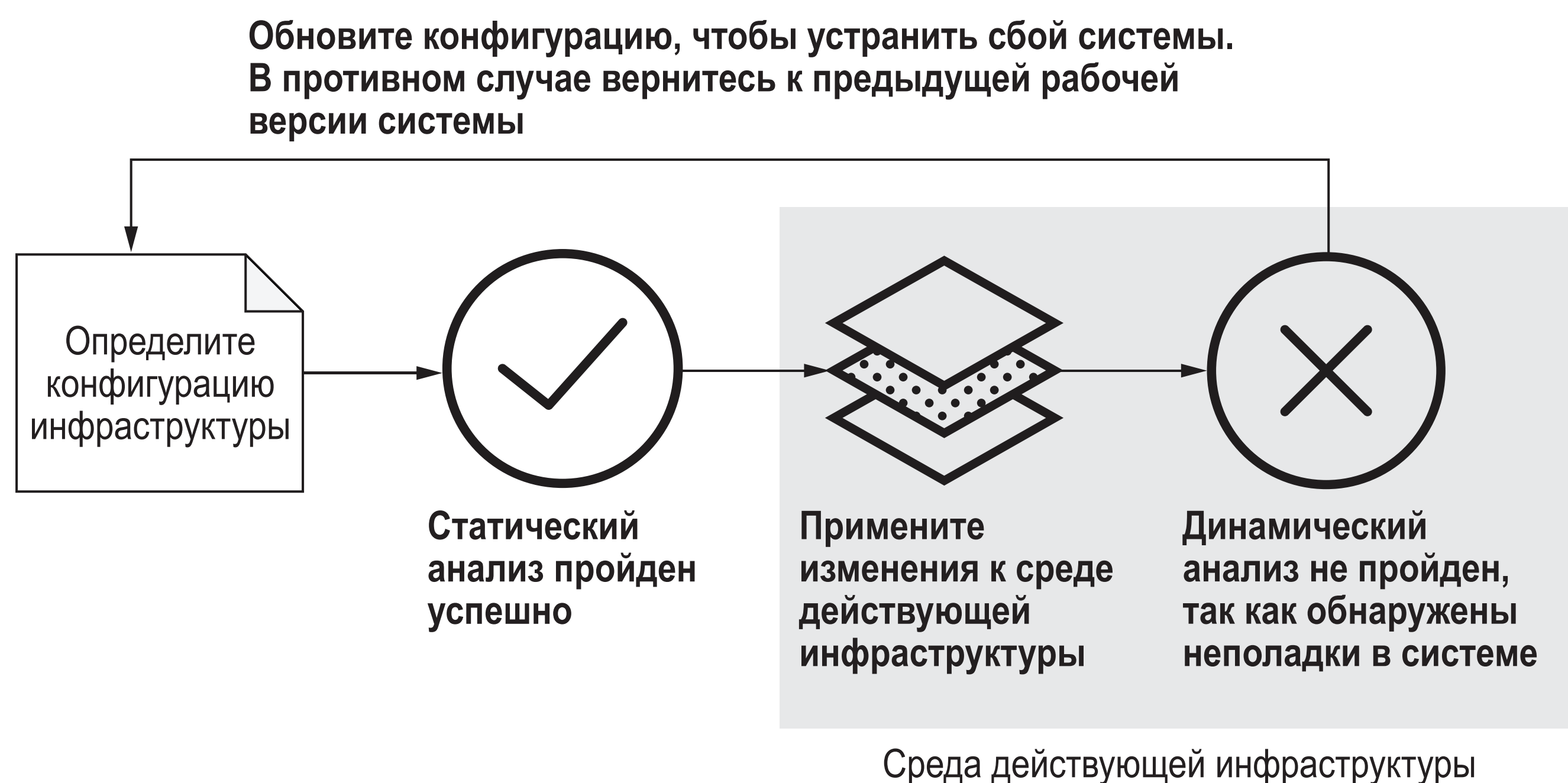


Рис. 6.4 Если динамический анализ провален, мы можем исправить среду тестирования, обновив конфигурацию или вернувшись к ранее работавшей конфигурации

Обратите внимание, что для создания сервера и проверки его подключения вашему сценарию тестирования необходима действующая сеть. Тесты, проверяющие функциональность инфраструктуры после применения изменений к действующим ресурсам инфраструктуры, выполняют динамический анализ.

ОПРЕДЕЛЕНИЕ Динамический анализ IaC проверяет функциональность системы после применения изменений к действующим ресурсам инфраструктуры.

Если динамические тесты пройдены успешно, у нас будет больше уверенности в том, что обновление не содержит ошибок. Если тесты не пройдены, нужно исправить конфигурацию или сценарии и перезапустить тесты. Набор динамических тестов образует систему раннего предупреждения о потенциально опасных изменениях, которые могут привести к нарушению ресурсов инфраструктуры и функциональности системы.

Но сейчас мы говорили о динамическом анализе действующей среды. Как быть, если мы не знаем, будет ли вообще работать среда после обновления? Можем ли мы изолировать эти тесты от про-

мышленной среды? Вместо того чтобы применять все изменения к промышленной среде и тестировать ее постфактум, можно создать промежуточную среду тестирования, чтобы отделить обновления и протестировать их.

6.1.3 Среда тестирования инфраструктуры

Некоторые организации дублируют в отдельной среде целые сети, чтобы можно было тестировать полномасштабные обновления. Наличие специальной тестовой среды позволяет выявлять и устранять недостатки конфигурации и вносить изменения, не затрагивая критически важные для бизнеса системы.

Когда перед переходом в активную среду мы запускаем тесты в отдельной среде, тем самым мы добавляем цикл тестирования инфраструктуры. На рис. 6.5 показано, что мы сохраняем этап статического анализа. Однако мы применяем изменения сети в тестовой среде и запускаем для нее динамический анализ. Если обновления не вызывают проблем в тестовой среде, мы применяем изменения в промышленной среде и запускаем динамический анализ еще раз.

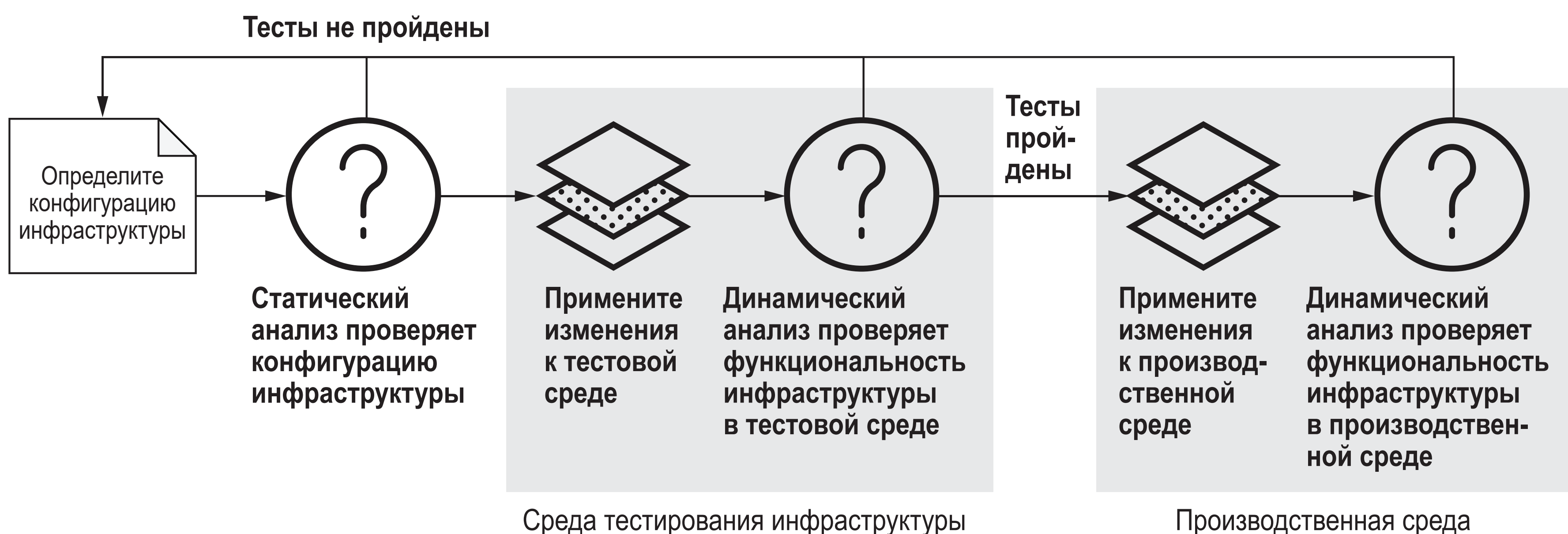


Рис. 6.5 Перед применением изменений в промышленной среде мы выполняем статический и динамический анализы в тестовой среде

Тестовая среда изолирует изменения и тесты от промышленной среды.

ОПРЕДЕЛЕНИЕ *Тестовая среда* (testing environment) отделена от промышленной и применяется для тестирования изменений инфраструктуры.

Пробный прогон в тестовой среде поможет вам попрактиковаться и проверить изменения перед развертыванием в промышленной среде. Вы лучше поймете, как они влияют на существующие системы. Если не получается исправить обновления, можно вернуть тестовую среду к последней рабочей конфигурации. Тестовые среды обычно применяют для следующих целей:

- изучение эффекта обновления инфраструктуры перед его применением в производственной системе;
- изолированное тестирование модулей инфраструктуры (о практике совместного использования модулей говорилось в главе 5).

Однако имейте в виду, что вам необходимо заботиться о тестовой среде наравне с производственной. По возможности среда тестирования инфраструктуры должна соответствовать следующим требованиям:

- ее конфигурация должна быть максимально похожа на производственную;
- это должна быть среда, отдельная от среды разработки приложения;
- она должна быть постоянной (т. е. не создается и не удаляется при каждом тестировании).

В предыдущих главах я упоминала о важности уменьшения дрейфа между разными средами. Если ваша среда тестирования инфраструктуры полностью повторяет промышленную среду, тестирование будет более точным. Кроме того, тестирование должно происходить отдельно от среды, предназначенной для разработки приложений. Убедившись, что изменения в вашей инфраструктуре ничего не нарушают, вы можете перенести их в среду разработки приложения.

Соблюдение этих принципов помогает иметь постоянную среду тестирования инфраструктуры. Так, вы сможете проверить, повлияют ли обновления работающей инфраструктуры на критически важные для бизнеса системы. К сожалению, поддержка отдельной среды тестирования инфраструктуры может оказаться непрактичной с точки зрения затрат или ресурсов. В главе 12 я опишу некоторые методы управления затратами на среду тестирования.

В оставшейся части этой главы я расскажу о различных типах тестов, выполняющих статический и динамический анализы, и о том, как они вписываются в вашу среду тестирования. Некоторые тесты позволят вам снизить зависимость от среды тестирования. Другие будут иметь решающее значение для оценки функциональности промышленной системы после изменений. В главе 11 я рассматриваю методы отката, специфичные для промышленной эксплуатации, и интегрирую тестирование в непрерывную поставку инфраструктуры.

6.2 Модульные тесты

В начале главы я упоминала важность проведения статического анализа IaC. Статический анализ оценивает файлы конкретных конфигураций. Какие тесты мы могли бы применить для статического анализа?

Представьте, что у вас есть модуль фабрики для создания сети с именем `hello-world-network` и трех подсетей с диапазоном IP-ад-

ресов 10.0.0.0/16. Вы хотите проверить их сетевые имена и диапазоны IP-адресов. Вы ожидаете, что подсети разделят между собой диапазон 10.0.0.0/16.

Мы можем написать тесты для проверки диапазонов имен сети и IP-адресов подсети в IaC без создания сети и подсети. Этот статический анализ проверяет параметры конфигурации на предмет ожидаемых значений за считанные секунды.

На рис. 6.6 показано, что наш статический анализ состоит из нескольких тестов, выполняемых одновременно. Мы проверяем имя сети, количество подсетей и диапазоны IP-адресов для подсетей.

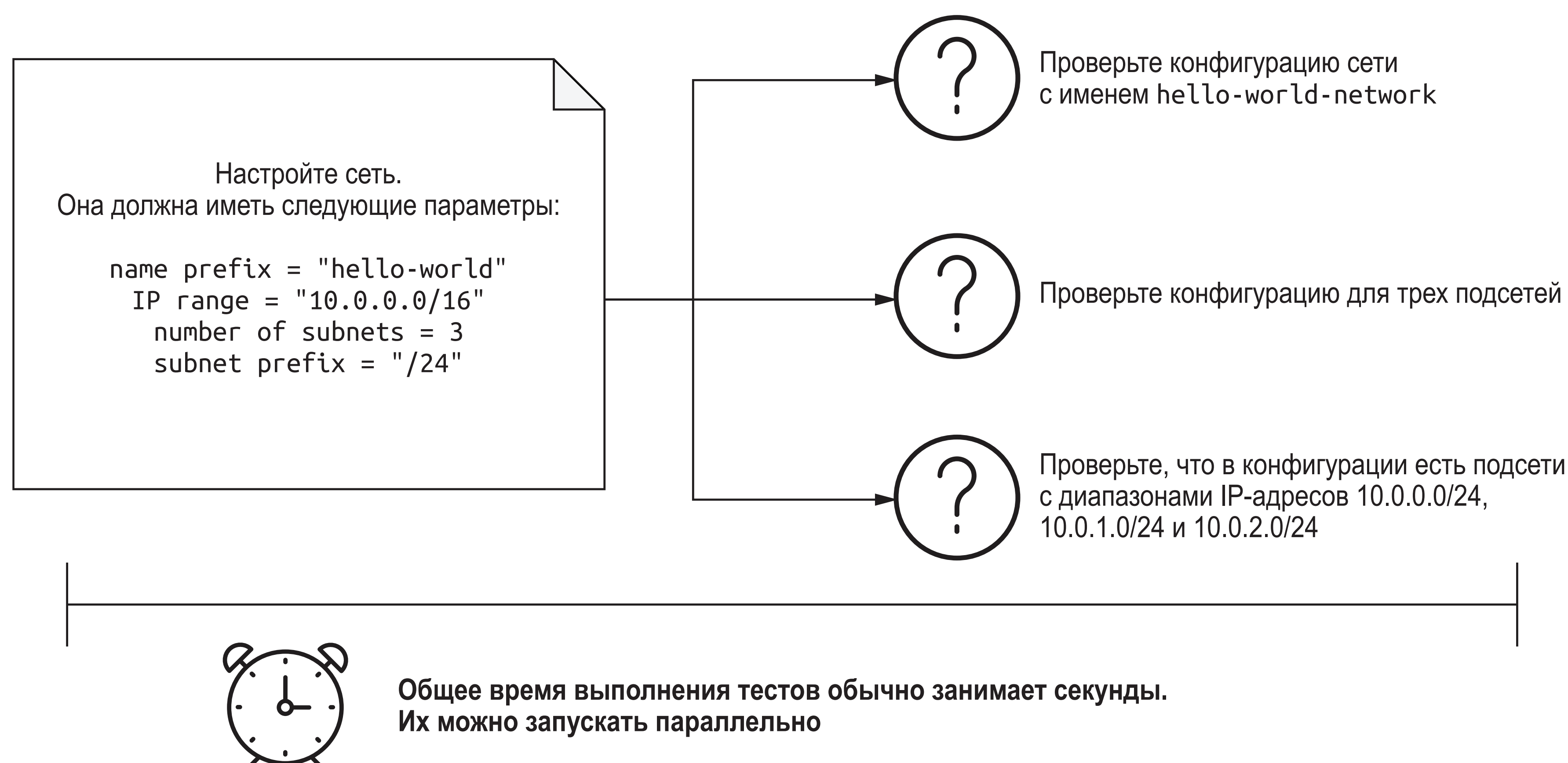


Рис. 6.6 Модульные тесты проверяют, что параметр конфигурации, например имя сети, соответствует ожидаемому значению

Таким образом, мы провели модульные тесты в сети IaC. Модульный тест выполняется изолированно и статически анализирует конфигурацию или состояние инфраструктуры. Эти тесты не рассматривают активные ресурсы или зависимости инфраструктуры и проверяют минимальное подмножество конфигурации.

ОПРЕДЕЛЕНИЕ *Модульные тесты* (unit test) статически анализируют конфигурацию или состояние инфраструктуры в виде открытого текста. Они не рассматривают активные ресурсы или зависимости инфраструктуры.

Модульные тесты могут анализировать метаданные в файлах конфигурации или состояния инфраструктуры. Некоторые инструменты предоставляют информацию непосредственно в конфигурации, тогда как другие предоставляют значения через состояние. В следующих нескольких разделах мы рассмотрим примеры тестирования обоих типов файлов. В зависимости от применяемого инструмента

IaC, среды тестирования и ваших предпочтений может выполняться тестирование как одного из двух типов, так и обоих сразу.

6.2.1 Тестирование конфигурации инфраструктуры

Мы начнем с написания модульных тестов для модулей, которые используют паттерны для создания конфигурации инфраструктуры. Наш модуль фабрики сетей использует функцию для создания объекта с конфигурацией сети. Нас интересует, генерирует ли функция `_network_configuration` правильную конфигурацию.

Для проверки модуля фабрики сетей мы напишем модульный тест, который проверяет функции, генерирующие конфигурацию JSON для сетей и подсетей. Модульный тест проверяет три показателя: имя сети, количества подсетей и диапазон IP-адресов.

Фреймворк Pytest будет находить тесты, выполняя поиск файлов и тестов с префиксом `test_`. В листинге 6.1 мы назвали тестовый файл `test_network.py`, чтобы фреймворк мог его найти. Каждый тест в файле имеет префикс `test_` и описательную информацию о том, что проверяет тест.

Листинг 6.1 Использование pytest для запуска модульных тестов в `test_network.py`

```
import pytest 1
from main import NetworkFactoryModule 2

NETWORK_PREFIX = 'hello-world' 3
NETWORK_IP_RANGE = '10.0.0.0/16' 3

@pytest.fixture(scope="module") 4
def network(): 4
    return NetworkFactoryModule( 4
        name=NETWORK_PREFIX, 4
        ip_range=NETWORK_IP_RANGE, 4
        number_of_subnets=3) 4

@pytest.fixture 5
def network_configuration(network): 5
    return network._network_configuration()['google_compute_network'][0] 5

@pytest.fixture 6
def subnet_configuration(network): 6
    return network._subnet_configuration()[ 6
        'google_compute_subnetwork'] 6

def test_configuration_for_network_name(network, network_configuration): 7
    assert network_configuration[network._network_name][ 7
        0]['name'] == f"{NETWORK_PREFIX}-network"
```



```
def test_configuration_for_three_subnets(subnet_configuration):
    assert len(subnet_configuration) == 3

def test_configuration_for_subnet_ip_ranges(subnet_configuration):
    for i, subnet in enumerate(subnet_configuration):
        assert subnet[next(iter(subnet))
                       ][0]['ip_cidr_range'] == f"10.0.{i}.0/24"
```

- ❶ Импортируем `pytest` – библиотеку тестирования Python. Имена тестовых файлов должны содержать префикс `test_`, чтобы фреймворк мог их найти и запустить.
- ❷ Импортируем модуль фабрики сетей из `main.py`. Вам необходимо запустить метод настройки сети.
- ❸ Задаем константы ожидаемых значений, такие как префикс сети и диапазон IP-адресов.
- ❹ Создаем сеть как объект тестирования на основе ожидаемых значений. Мы используем общий тестовый стенд для выполнения всех тестов.
- ❺ Создаем отдельный тестовый объект для конфигурации сети, поскольку нам необходимо проанализировать `google_compute_network`. Один из тестов использует его для проверки имени сети.
- ❻ Создаем отдельный тестовый объект для конфигурации подсети, поскольку нам необходимо проанализировать `google_compute_subnetwork`. Два теста используют его для проверки количества подсетей и диапазонов их IP-адресов.
- ❼ `Pytest` запустит этот тест, чтобы проверить конфигурацию имени сети на соответствие `hello-world-network`. Он использует тестовый объект `network_configuration`.
- ❽ `Pytest` запустит этот тест, чтобы проверить конфигурацию на наличие количества подсетей, равного 3. Он использует тестовый объект `subnet_configuration`.
- ❾ `Pytest` проверит правильность диапазона IP-адресов подсети в примере конфигурации сети. Он использует тестовый объект `subnet_configuration`.

Эквиваленты AWS и Azure

Чтобы перенести код листинга 6.1 в AWS, используйте ресурс Terraform `aws_subnet` (<http://mng.bz/J2vZ>) и получите значение атрибута `cidr_block`.

Для Azure используйте ресурс Terraform `azurerm_subnet` (<http://mng.bz/wo05>) и получите значение атрибута `address_prefixes`.

Файл теста содержит статический сетевой объект, передаваемый между тестами. Этот *тестовый стенд* создает согласованный сетевой объект, на который может ссылаться каждый тест. Таким образом мы уменьшаем количество повторяющегося кода, используемого для создания тестового ресурса.

ОПРЕДЕЛЕНИЕ *Тестовый стенд* (test fixture) – это известная конфигурация, используемая для запуска теста. Она часто содержит известные или ожидаемые значения для данного ресурса инфраструктуры. Некоторые стенды отдельно анализируют информацию о сети и подсети. Каждый раз, когда мы добавляем новые тесты, нам не нужно копировать

и вставлять анализ. Вместо этого мы ссылаемся на тестовый стенд конфигурации.

Вы можете запустить `pytest` из командной строки и передать аргумент с тестовым файлом. `Pytest` запускает набор из трех тестов и выводит результаты их успеха:

```
$ pytest test_network.py
===== test session starts =====
collected 3 items

test_network.py ... [100%]

===== 3 passed in 0.06s =====
```

В этом примере мы импортируем модуль сетевой фабрики, создаем сетевой объект с конфигурацией и тестируем его. Вам не нужно записывать какую-либо конфигурацию в файл. Вместо этого мы ссылаемся на функцию и тестируем объект.

Этот пример основан на том же подходе, который я применяю к коду приложения для модульного тестирования. Зачастую это приводит к созданию более мелких модульных функций, которые можно тестировать более эффективно.

Функция, генерирующая конфигурацию сети, должна вывести конфигурацию для теста. В противном случае тесты не смогут извлекать и сравнивать значения.

6.2.2 Тестирование с помощью предметно-ориентированных языков

Как проверить конфигурацию сети и подсети, если вы используете DSL? У вас нет функций, которые вы можете вызвать в своем тесте. Вместо этого ваши модульные тесты должны извлекать значения из файла конфигурации или пробного прогона. Оба типа файлов хранят своего рода метаданные в виде открытого текста о ресурсах инфраструктуры.

Представьте, что вы используете для создания своей сети DSL вместо Python. В следующем примере создается файл JSON с конфигурацией, совместимой с Terraform. Файл JSON содержит все три подсети, их диапазоны IP-адресов и имена. На рис. 6.7 показан запуск модульных тестов для файла конфигурации JSON сети. Тесты выполняются быстро, поскольку сети не развертываются.

Как правило, вы всегда можете выполнить модульное тестирование файлов, которые вы использовали для определения IaC. Если инструмент использует файл конфигурации, например CloudFormation, Terraform, Vicer, Ansible, Puppet, Chef и т. д., вы можете выполнить модульное тестирование любых строк конфигурации.

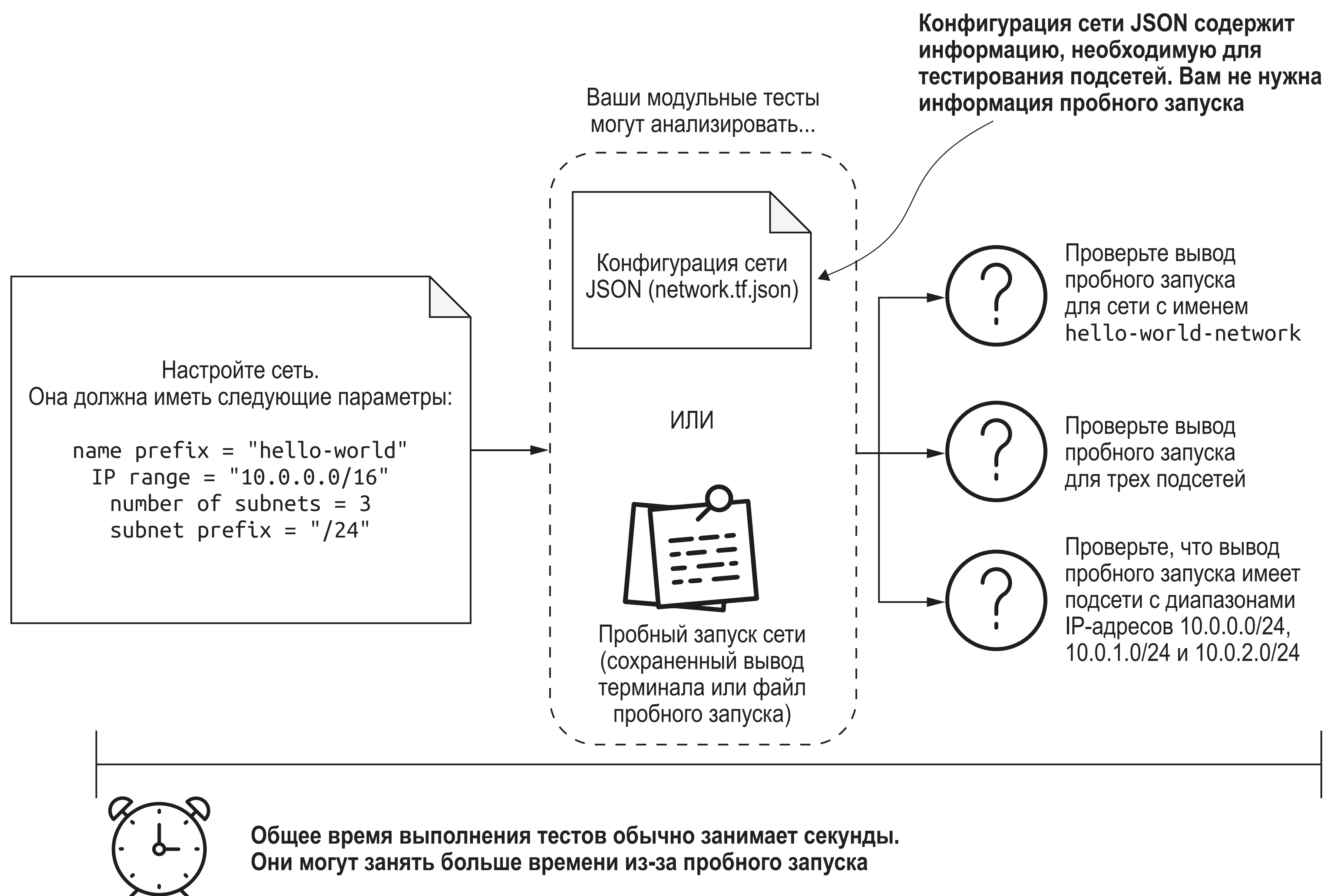


Рис. 6.7 Модульные тесты при пробных прогонах требуют создания предварительного просмотра изменений в ресурсах инфраструктуры и проверки допустимых параметров

В листинге 6.2 выполняется тестирование имени сети, количества подсетей и диапазонов IP-адресов подсети для нашего сетевого модуля без пробного прогона. Я запускаю аналогичные тесты с помощью `pytest`, чтобы проверить те же параметры.

Листинг 6.2 Использование `pytest` для запуска модульных тестов в `test_network_configuration.py`

```
import json 1
import pytest

NETWORK_CONFIGURATION_FILE = 'network.tf.json' 2

expected_network_name = 'hello-world-network' 3

@pytest.fixture(scope="module") 4
def configuration(): 4
    with open(NETWORK_CONFIGURATION_FILE, 'r') as f: 4
        return json.load(f) 4

@pytest.fixture 5
def resource(): 5
    def _get_resource(configuration, resource_type): 5
        for resource in configuration['resource']: 5
            if resource_type in resource.keys(): 5
```



```

        return resource[resource_type]
    return _get_resource

@pytest.fixture
def network(configuration, resource):
    return resource(configuration, 'google_compute_network')[0]

@pytest.fixture
def subnets(configuration, resource):
    return resource(configuration, 'google_compute_subnetwork')

def test_configuration_for_network_name(network):
    assert network[expected_network_name][0]['name'] \
        == expected_network_name

def test_configuration_for_three_subnets(subnets):
    assert len(subnets) == 3

def test_configuration_for_subnet_ip_ranges(subnets):
    for i, subnet in enumerate(subnets):
        assert subnet[next(iter(subnet))
            ][0]['ip_cidr_range'] == f"10.0.{i}.0/24"

```

- ❶ Импортируем библиотеку JSON Python, поскольку нам нужно будет загрузить файл JSON.
- ❷ Задаем константу с ожидаемым именем файла для конфигурации сети. Тесты считывают конфигурацию сети из network.tf.json.
- ❸ Задаем ожидаемое имя сети hello-world-network.
- ❹ Открываем файл JSON с конфигурацией сети и загружаем его в качестве тестового стенда.
- ❺ Создаем новый тестовый стенд, который получает загруженную конфигурацию JSON и извлекает информацию о ресурсах. Он анализирует JSON, исходя из структуры ресурсов JSON Terraform.
- ❻ Получаем ресурс Terraform google_compute_network из файла JSON.
- ❼ Получаем ресурс Terraform google_compute_subnetwork из файла JSON.
- ❽ Pytest запустит этот тест, чтобы проверить конфигурацию имени сети на соответствие hello-world-network. В проверке задействован тестовый стенд для сетей.
- ❾ Pytest запустит этот тест, чтобы проверить конфигурацию на наличие количества подсетей, равного 3. В проверке задействован тестовый стенд для подсетей.
- ❿ Pytest проверит правильную конфигурацию диапазона IP-адресов подсети. В проверке задействован тестовый стенд для подсетей.

Эквиваленты AWS и Azure

Чтобы перенести код листинга 6.2 в AWS, используйте ресурс Terraform aws_subnet (<http://mng.bz/J2vZ>) и получите значение атрибута cidr_block.

Для Azure используйте ресурс Terraform azure_rm_subnet (<http://mng.bz/wo05>) и получите значение атрибута address_prefixes.

Вы могли заметить, что модульные тесты для DSL похожи на тесты для обычных языков программирования. Они проверяют имя сети, количество подсетей и IP-адреса. Некоторые инструменты имеют специализированные среды тестирования. Обычно они используют один и тот же рабочий процесс: выполнение пробного прогона или генерация файла состояния и его анализ на предмет значений.

Однако ваш файл конфигурации может содержать не всю нужную информацию. Например, у вас не будет определенных конфигураций в Terraform или Ansible до тех пор, пока вы не выполните пробный прогон. Пробный прогон позволяет просмотреть изменения IaC без их развертывания, а также выявить и устранить потенциальные проблемы.

ОПРЕДЕЛЕНИЕ *Пробный прогон* (dry run) позволяет просмотреть изменения IaC без их развертывания. Он выявляет и решает потенциальные проблемы путем внутреннего анализа.

Пробные прогоны бывают разных форматов и стандартов. Большинство пробных прогонов выводят данные в терминал, и вы можете сохранить результат в файл. Некоторые инструменты автоматически создают файл пробного прогона.

Создание пробных прогонов для модульных тестов

Одни инструменты сохраняют результаты пробного прогона в файле, а другие выводят изменения в терминал. Если вы используете Terraform, вы записываете план Terraform в файл JSON, используя следующую команду:

```
$ terraform plan -out=dry_run && terraform show -json dry_run > dry_run.json
```

AWS CloudFormation использует понятие набора изменений, и вы можете проанализировать описание набора изменений после его завершения. Аналогично вы можете получить информацию о пробном прогоне Kubernetes с помощью опции `-dry-run=client` команды `kubectl run`.

Обычно я отдаю приоритет тестам, проверяющим файлы конфигурации. Я пишу тесты для анализа пробных прогонов, когда не могу получить значение из файлов конфигурации. Пробный прогон обычно требует сетевого доступа к API провайдера инфраструктуры и занимает некоторое время. Иногда выходные данные или файл содержат конфиденциальную информацию либо идентификаторы, которые тест не должен анализировать в явном виде.

Хотя конфигурация пробного прогона может не совпадать с традиционным определением модульных тестов, применяемых при

разработке программного обеспечения, анализ пробных прогонов не требует каких-либо изменений в активной инфраструктуре. Он остается формой статического анализа. Пробный прогон сам по себе служит модульным тестом для проверки ожидаемого поведения изменения перед его применением и вывода результатов.

6.2.3 Когда следует применять модульные тесты?

Модульные тесты помогают убедиться, что ваша логика генерирует правильные имена, создает правильное количество ресурсов инфраструктуры и вычисляет правильные диапазоны IP-адресов или другие атрибуты. Некоторые модульные тесты могут пересекаться с форматированием и линтингом – концепциями, о которых я упоминала в главе 2. Я отношу линтинг и форматирование к части модульного тестирования, поскольку они помогают нам понять, как называть и организовывать конфигурацию.

На рис. 6.8 показаны некоторые варианты использования модульных тестов. Вам следует написать дополнительные модульные тесты для проверки любой логики, которую вы использовали для создания конфигурации инфраструктуры, особенно с циклами или условными операторами if-else. Модульные тесты также могут выявить неправильные или проблемные конфигурации, например неправильную операционную систему.

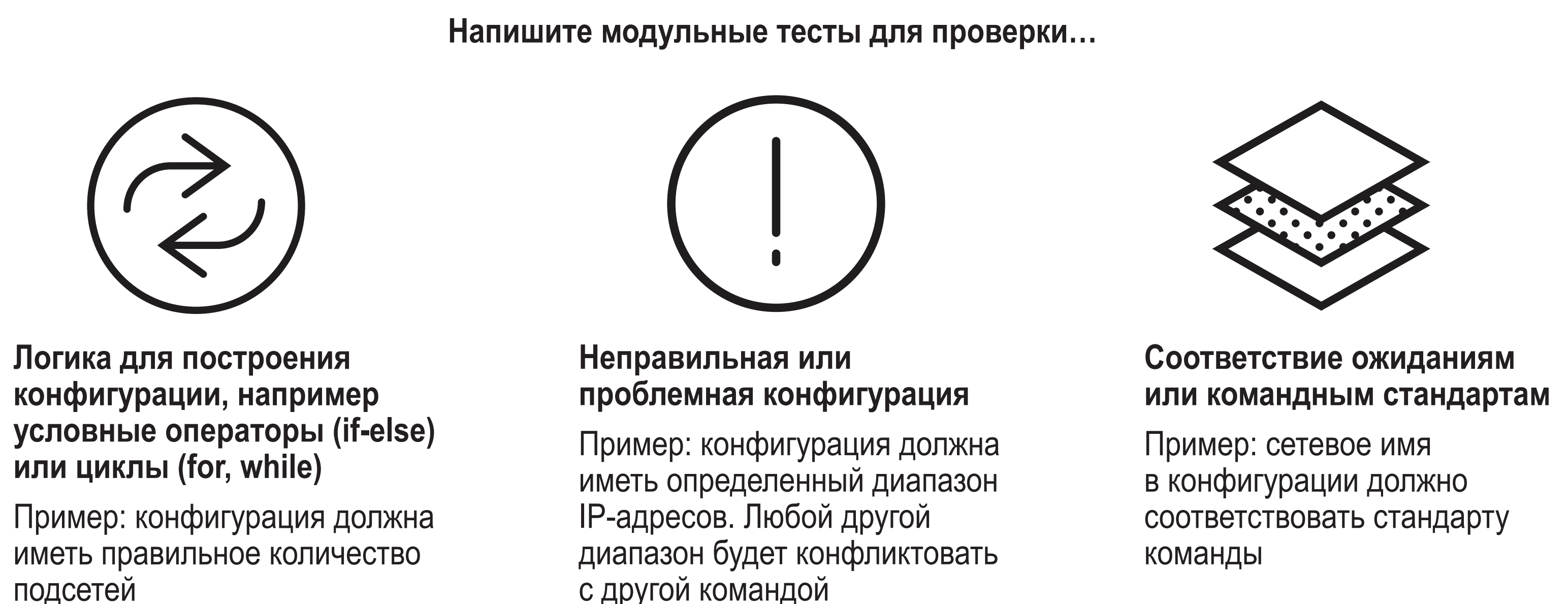


Рис. 6.8 Напишите модульные тесты для проверки логики ресурса, выявления потенциальных проблем или определения стандартов команды

Поскольку модульные тесты проверяют конфигурацию изолированно, они не в полной мере отражают влияние обновлений на систему. Поэтому не следует ожидать, что модульное тестирование предотвратит серьезный сбой во время изменений в промышленной среде. Однако применять модульные тесты все равно следует! Хотя модульные тесты не выявляют проблемы после внесения изменений, они могут предотвратить создание проблемных конфигураций еще до развертывания в промышленной среде.

Например, кто-то может случайно запустить конфигурацию для 1000 серверов вместо 10. Тест для проверки максимального количества серверов в конфигурации может предотвратить чрезмерную нагрузку на инфраструктуру и снизить затраты. Модульные тесты также могут предотвратить любую небезопасную или несовместимую конфигурацию инфраструктуры в промышленной среде. В главе 8 я расскажу, как применять модульные тесты для защиты и аудита конфигурации инфраструктуры.

Помимо раннего выявления неправильных значений конфигурации, модульные тесты помогают автоматизировать проверку сложных систем. Если у вас много ресурсов инфраструктуры, управляемых разными командами, вы больше не сможете вручную выполнять поиск по одному списку ресурсов и проверять каждую конфигурацию. Модульные тесты передают другим командам наиболее важные или стандартные конфигурации. Когда вы пишете модульные тесты для модулей инфраструктуры, следует убедиться, что внутренняя логика модуля создает ожидаемые ресурсы.

Модульное тестирование средств автоматизации

Для описания хороших модульных тестов требуется целая книга! Я ограничила свои объяснения в этом разделе тестированием конфигурации инфраструктуры. Однако вы можете написать собственный инструмент автоматизации, напрямую обращающийся к API инфраструктуры. Автоматизация использует более последовательный подход для пошаговой настройки ресурса (так называемый *императивный стиль*).

Применяйте модульные тесты для проверки отдельных шагов и их идемпотентности. Модульные тесты должны запускать отдельные шаги с различными предварительными условиями и проверять, что они дают одинаковый результат. Если вам нужен доступ к API инфраструктуры, имитируйте ответы API в своих модульных тестах.

К разновидностям модульных тестов можно отнести проверку того, что вы создали ожидаемое количество ресурсов инфраструктуры, закрепили определенные версии инфраструктуры или использовали правильный стандарт именования. Модульные тесты выполняются быстро и обеспечивают быструю обратную связь практически при нулевых затратах (после того, как вы их написали!). Они выполняются в течение нескольких секунд, поскольку не публикуют обновления и не требуют создания активных ресурсов инфраструктуры. Во времени выполнения модульных тестов для проверки результатов пробного прогона добавляется немного времени, затраченного на сам пробный прогон.

6.3 Контрактные тесты

Модульные тесты проверяют конфигурацию или модули по отдельности, но как насчет зависимостей между модулями? В главе 4 я упомянула идею контракта между зависимостями. Выходные данные одного модуля должны совпадать с ожидаемыми входными данными другого модуля. Соблюдение этого соглашения тоже можно протестировать.

Допустим, мы решили создать сервер в сети. Сервер получает доступ к сетевому имени и IP-адресу, используя фасад, который отражает имя и диапазон IP-адресов сети. Как узнать, что сетевой модуль выводит имя сети и диапазон IP CIDR, а не другой идентификатор или конфигурацию?

На рис. 6.9 показано, как мы используем контрактный тест, чтобы проверить, правильно ли сетевой модуль выводит фасад. Фасад должен содержать имя сети и диапазон IP-адресов. Если тест не пройден, это показывает, что сервер не может быть создан.

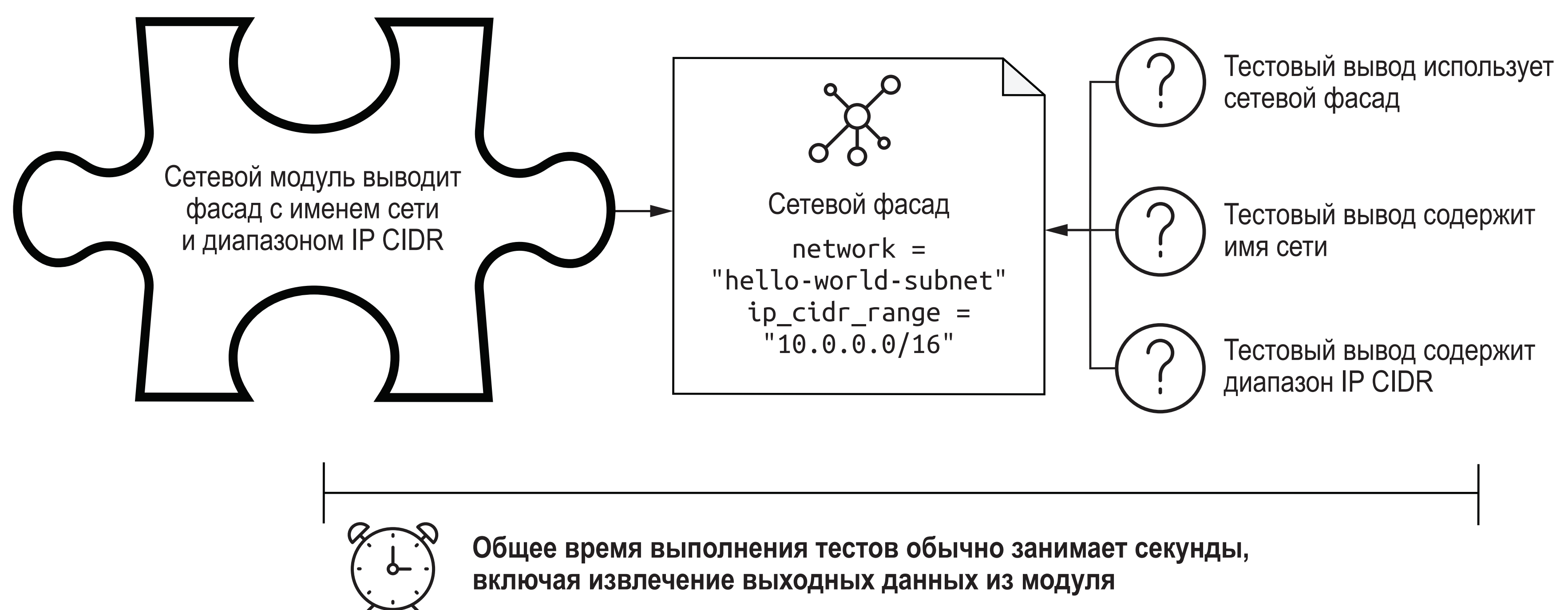


Рис. 6.9 Контрактные тесты позволяют быстро проверить, что параметр конфигурации соответствует ожидаемому значению, например сетевой фасад выводит правильные данные

Контрактный тест использует статический анализ для проверки соответствия входных и выходных данных модуля ожидаемому значению или формату.

ОПРЕДЕЛЕНИЕ *Контрактные тесты* (contract test) выполняют статический анализ и сравнивают входные и выходные данные модуля или ресурса на предмет соответствия ожидаемому значению или формату.

Регулярное выполнение контрактных тестов обеспечивает возможность развития отдельных модулей при сохранении интеграции

между ними. Если у вас много зависимостей инфраструктуры, вы не можете вручную проверить все их общие атрибуты. Контрактный тест автоматизирует проверку типа и значения атрибутов между модулями.

Контрактные тесты наиболее полезны при проверке входных и выходных данных сильно параметризованных модулей (таких как паттерны фабричный, «прототип» или «строитель»). Разработка и выполнение контрактных тестов помогают обнаружить неправильные входные и выходные данные и документировать минимальные ресурсы модуля. Если у вас нет контрактных тестов для ваших модулей, вы не узнаете, что своими действиями сломали что-то в системе, до тех пор, пока в очередной раз не примените конфигурацию к реальной среде.

В листинге 6.3 показан пример реализации контрактного теста. Используя `pytest`, мы настраиваем тест, создавая сеть с фабричным модулем. Затем мы убеждаемся, что выходные данные сети содержат фасадный объект с именем сети и диапазоном IP-адресов. Мы добавляем эти тесты к модульным тестам сервера.

Листинг 6.3 Контрактный тест для сравнения выходных данных одного модуля с требованиями к входным данным другого модуля

```
from network import NetworkFactoryModule, NetworkFacade
import pytest

network_name = 'hello-world'
network_cidr_range = '10.0.0.0/16'

@pytest.fixture
def network_outputs():
    network = NetworkFactoryModule(
        name=network_name,
        ip_range=network_cidr_range)
    return network.outputs()

def test_network_output_is_facade(network_outputs):
    assert isinstance(network_outputs, NetworkFacade)

def test_network_output_has_network_name(network_outputs):
    assert network_outputs._network == f"{network_name}-subnet"

def test_network_output_has_ip_cidr_range(network_outputs):
    assert network_outputs._ip_cidr_range == network_cidr_range
```

- ❶ Pytest запустит этот тест, чтобы проверить, соответствует ли имя сети ожидаемому значению `hello-world`.
- ❷ Pytest запустит этот тест, чтобы убедиться, что диапазон IP CIDR сетевого вывода соответствует `10.0.0.0/16`.
- ❸ Настраиваем тестовый стенд, который использует модуль сетевой фабрики и возвращает его выходные данные.

- ④ Создаем сеть с использованием модуля фабрики с именем и диапазоном IP-адресов.
- ⑤ Тестовый стенд должен возвращать сетевой фасад с различными выходными атрибутами.
- ⑥ Pytest запустит этот тест, чтобы проверить, выводит ли модуль объект сетевого фасада.
- ⑦ Pytest запустит этот тест, чтобы проверить, соответствует ли имя сети ожидаемому значению hello-world.
- ⑧ Pytest запустит этот тест, чтобы убедиться, что диапазон IP CIDR сетевого выхода соответствует 10.0.0.0/16.

Представьте, что после обновления сетевой модуль выводит идентификатор сети вместо имени. Это нарушает функциональность модуля восходящего сервера, поскольку сервер ожидает имя сети! Контрактное тестирование гарантирует, что вы не нарушите контракт (или интерфейс) между двумя модулями при обновлении любого из них. Используйте контрактный тест для проверки фасадов и адаптеров при реализации зависимостей между ресурсами.

Почему вам следует добавить контрактный тест на сервер – ресурс более высокого уровня? Ваш сервер ожидает определенных выходных данных из сети. Если сетевой модуль изменяется, то модуль верхнего уровня должен это заметить.

Иными словами, модуль верхнего уровня должен учитывать изменения в модуле нижнего уровня, чтобы сохранить возможность компоновки и развития. Вы не должны без крайней необходимости вносить существенные изменения в интерфейс модуля низкого уровня, поскольку это может повлиять на другие модули, которые от него зависят.

Предметно-ориентированные языки

В листинге 6.3 для проверки выходных данных модуля используется Python. Если вы применяете инструмент с DSL, вам могут быть доступны встроенные функции, позволяющие проверять соответствие входных данных определенным типам или регулярным выражениям (например, проверка допустимого идентификатора или форматирования имени). Если инструмент не имеет функции проверки, возможно, придется использовать отдельную тестовую среду для анализа типов выходных данных из конфигурации модуля нижнего уровня и сравнения их с типами входных данных модуля верхнего уровня.

Контрактные тесты инфраструктуры требуют определенного способа извлечения ожидаемых входных и выходных данных, например обращения к API провайдеров инфраструктуры и проверки ответов на ожидаемые значения. Иногда приходится создавать тестовые ресурсы для проверки параметров и хорошо понимать, как следует структурировать такие поля, как идентификатор. Когда

вам нужно выполнить вызовы API или создать временные ресурсы, контрактные тесты могут выполняться дольше, чем модульный тест.

6.4 Интеграционные тесты

Как узнать, что можно применить изменения конфигурации или модуля к инфраструктурной системе? Вам необходимо применить изменения в среде тестирования и *динамически* проанализировать работающую инфраструктуру. Интеграционный тест выполняется в тестовых средах для проверки успешности обновлений в модуле или конфигурации.

ОПРЕДЕЛЕНИЕ *Интеграционные тесты* (integration test) выполняют в средах тестирования и динамически анализируют ресурсы инфраструктуры, чтобы убедиться, что изменения модулей или конфигурации соответствуют ожиданиям.

Интеграционные тесты требуют наличия изолированной среды тестирования для проверки интеграции модулей и ресурсов. В следующих разделах вы узнаете об интеграционных тестах, которые можно написать для модулей и конфигураций инфраструктуры.

6.4.1 Тестирование модулей

Допустим, вы разработали модуль, создающий сервер GCP. Вы хотите убедиться, что можете успешно создать и обновить сервер, поэтому пишете интеграционный тест, как показано на рис. 6.10.

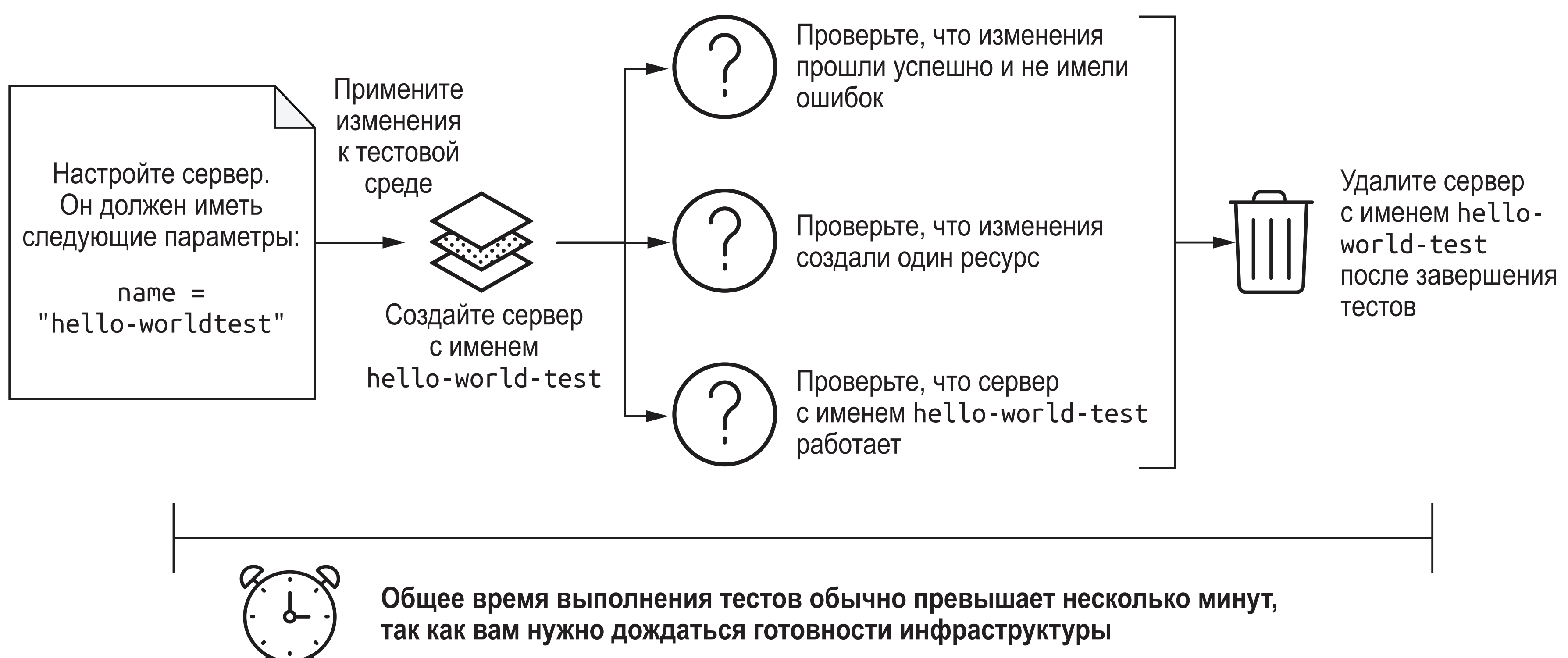


Рис. 6.10 Интеграционные тесты обычно создают и обновляют ресурсы инфраструктуры в среде тестирования, проверяют их конфигурацию и состояние на корректность или доступность и удаляют их после тестов

Сначала вы настраиваете сервер и применяете изменения в среде тестирования. Затем вы запускаете интеграционные тесты, чтобы убедиться, что обновление конфигурации прошло успешно, создаете сервер и называете его `hello-world-test`. Общее время выполнения теста занимает несколько минут, поскольку вам нужно дождаться готовности сервера.

При разработке интеграционного теста вам необходимо сравнить активный ресурс с вашим IaC. Активный ресурс сообщает вам, успешно ли развернут ваш модуль. Если кто-то не сможет развернуть модуль, он потенциально разрушит свою инфраструктуру.

Интеграционный тест должен получить информацию об активном ресурсе с помощью API провайдера инфраструктуры. Например, вы можете импортировать библиотеку Python для доступа к API GCP в интеграционном тесте вашего серверного модуля. Интеграционный тест импортирует библиотеку Python `Libcloud` в качестве клиентского SDK для API GCP.

Тест в листинге 6.4 создает конфигурацию сервера с помощью модуля, ожидает развертывания сервера и проверяет состояние сервера в GCP API. Если сервер возвращает статус `running`, тест пройден. В противном случае тест провален и выявлена проблема с модулем. Наконец, тест удаляет созданный им тестовый сервер.

Листинг 6.4 Интеграционные тесты для создания сервера в `test_integration.py`

```
from libcloud.compute.types import NodeState ①
from main import generate_json, SERVER_CONFIGURATION_FILE
import os
import pytest
import test_utils

TEST_SERVER_NAME = 'hello-world-test'

@pytest.fixture(scope='session') ②
def apply_changes(): ②
    generate_json(TEST_SERVER_NAME) ③
    assert os.path.exists(SERVER_CONFIGURATION_FILE) ③
    assert test_utils.initialize() == 0 ④
    yield test_utils.apply() ④
    assert test_utils.destroy() == 0 ⑤
    os.remove(SERVER_CONFIGURATION_FILE) ⑤

def test_changes_have_successful_return_code(apply_changes): ⑥
    return_code = apply_changes[0]
    assert return_code == 0

def test_changes_should_have_no_errors(apply_changes): ⑦
    errors = apply_changes[2]
    assert errors == b''
```



```
def test_changes_should_add_1_resource(apply_changes):
    output = apply_changes[1].decode(encoding='utf-8').split('\n')
    assert 'Apply complete! Resources: 1 added, ' + \
        '0 changed, 0 destroyed' in output[-2]

def test_server_is_in_running_state(apply_changes):
    gcp_server = test_utils.get_server(TEST_SERVER_NAME)
    assert gcp_server.state == NodeState.RUNNING
```

- ❶ Pytest использует Libcloud для вызова API GCP, получения текущего состояния сервера и проверки, что сервер работает.
- ❷ Во время сеанса тестирования мы используем тестовый стенд pytest для применения конфигурации и создаем тестовый сервер на GCP.
- ❸ Создаем файл Terraform JSON, который содержит серверный модуль.
- ❹ С помощью Terraform инициализируем и развертываем сервер, описанный в файле Terraform JSON.
- ❺ Удаляем тестовый сервер с помощью Terraform, а затем удаляем файл конфигурации JSON в конце тестового сеанса.
- ❻ Pytest запустит этот тест, чтобы убедиться, что выходной статус изменений сообщает об успешном выполнении.
- ❼ Pytest запустит этот тест, чтобы убедиться, что изменения не возвращают ошибку.
- ❽ Pytest запустит этот тест и проверит, добавлен ли в конфигурацию один ресурс – сервер.
- ❾ Pytest использует Libcloud для вызова API GCP, получения текущего состояния сервера и проверки, что сервер работает.

Эквиваленты AWS и Azure

Чтобы перенести код листинга 6.4, вам необходимо обновить IaC для создания экземпляра Amazon EC2 или виртуальной машины Azure Linux. Затем вам следует обновить драйвер Apache Libcloud для использования Amazon EC2 Driver (<http://mng.bz/qYex>) или драйвера вычислителя Azure ARM Compute Driver (<http://mng.bz/7yjQ>). Инициализация драйвера и IaC изменятся, а тест останется прежним.

Когда вы запустите тесты из этого файла в командной строке, вы заметите, что их выполнение занимает несколько минут, поскольку тестовый сеанс создает сервер и удаляет его:

```
$ pytest test_integration.py
===== test session starts =====
collected 4 items

test_integration.py .... [100%]

===== 4 passed in 171.31s (0:02:51) =====
```

В интеграционных тестах сервера применяются два основных метода. Сначала выполняется такая последовательность действий:

- ❶ рендеринг конфигурации, если применимо;

- 2 развертывание изменений в ресурсах инфраструктуры;
- 3 выполнение тестов с обращением к API провайдера инфраструктуры для сравнения;
- 4 удаление ресурсов инфраструктуры, если применимо.

В этом примере последовательность реализуется с использованием тестового стенда. Вы можете использовать его для применения любой произвольной конфигурации инфраструктуры и удаления ее после тестирования.

ПРИМЕЧАНИЕ Интеграционные тесты для инструментов управления конфигурацией работают аналогично. Например, вы можете устанавливать пакеты и запускать процессы на своем сервере. После запуска тестов вы дополнительно проверяете пакеты и процессы сервера и уничтожаете сервер. Вместо написания тестов с использованием языка программирования я рекомендую применять специализированный инструментарий для тестирования сервера, который подключается к серверу и запускает тесты системы.

Далее вы запускаете интеграционные тесты модулей в отдельной среде тестирования модулей (например, в тестовой учетной записи или проекте) отдельно от тестовых или промышленных сред, поддерживающих приложения. Чтобы предотвратить конфликты с другими тестами модулей в среде, вы помечаете и именуете ресурсы на основе конкретного типа модуля, версии или хеша коммита.

ОПРЕДЕЛЕНИЕ *Среда тестирования модулей* (module-testing environment) отделена от промышленной среды и предназначена для тестирования изменений модулей.

Тестирование модулей в среде, отличной от общей тестовой или промышленной среды, помогает изолировать неисправные модули от активной среды с приложениями. С помощью тестирования модулей вы также можете измерять и контролировать затраты на инфраструктуру. В главе 12 более подробно рассматривается стоимость облачных вычислений.

6.4.2 Тестирование конфигурации среды

Интеграционные тесты для модулей инфраструктуры могут создавать и удалять ресурсы в тестовой среде, но интеграционные тесты конфигураций среды так не работают. Представьте, что нам нужно добавить A-запись к вашему текущему доменному имени, настроенному с помощью составной или одиночной конфигурации. Как написать интеграционные тесты, чтобы проверить правильность добавления записи?

Мы столкнулись с двумя проблемами. Во-первых, мы не можем просто создавать, а затем удалять записи DNS в рамках интеграционных тестов, поскольку это может повлиять на приложения. Во-вторых, для настройки домена и добавления A-записи требуется существующий IP-адрес сервера.

Вместо создания и удаления сервера и A-записи в тестовой среде мы запускаем интеграционные тесты в *постоянной* (persistent) среде, полностью совпадающей с производственной средой. На рис. 6.11 показано, что мы обновляем запись DNS в IaC тестовой среды. Интеграционные тесты проверяют, соответствует ли DNS в тестовой среде ожидаемой правильной записи DNS. После прохождения теста вы можете обновить запись DNS для промышленной среды.

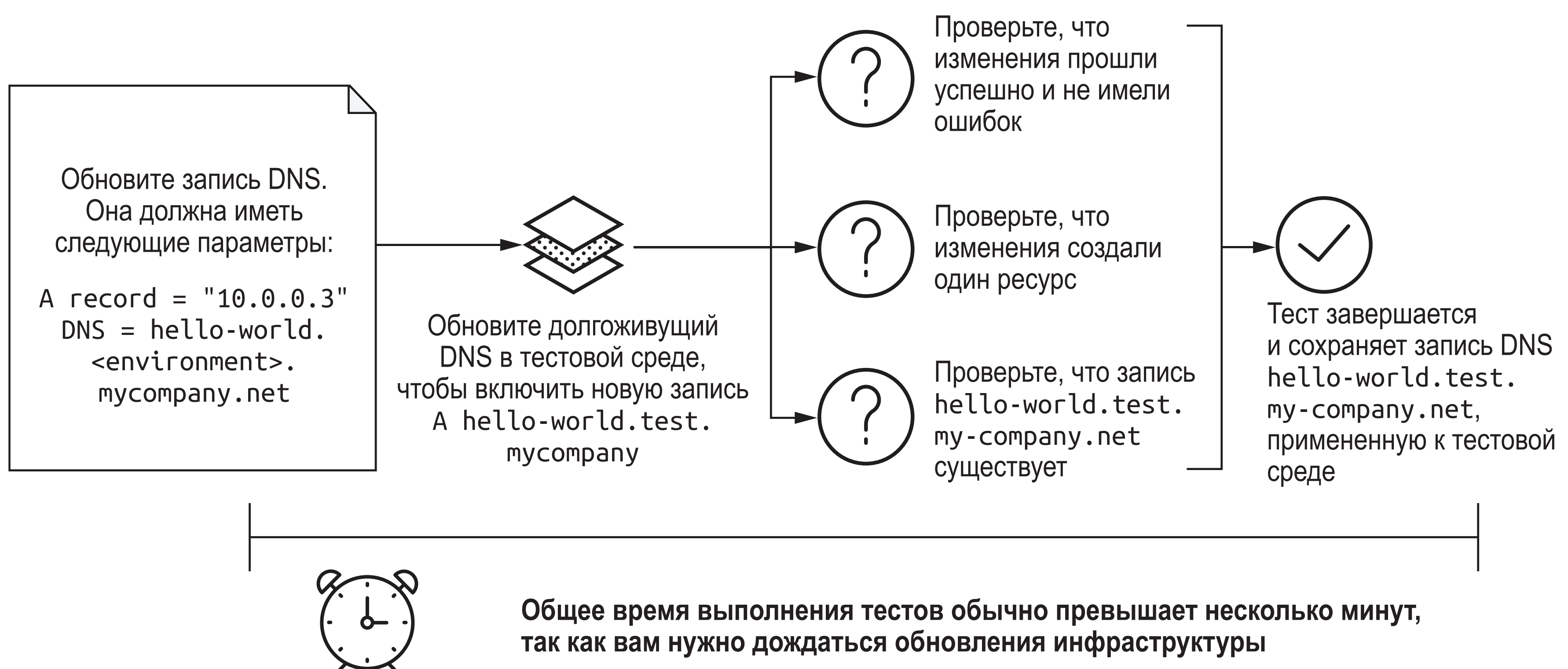


Рис. 6.11 Вы можете запустить интеграционные тесты в тестовой среде с долгоживущими ресурсами, чтобы изолировать изменения от производственной среды и уменьшить количество зависимостей, которые необходимо создать для теста

Зачем запускать тест DNS в постоянной тестовой среде? Во-первых, создание новой тестовой среды может занять много времени. Как ресурс верхнего уровня, DNS зависит от многих ресурсов нижнего уровня. Во-вторых, вам нужно иметь точное представление о том, как поведут себя изменения, прежде чем обновлять рабочую версию.

Тестовая среда охватывает подмножество зависимостей и сложностей промышленной системы, что позволяет проверить, работает ли ваша конфигурация как ожидается. Поддержание сходства сред означает, что изменения, протестированные в тестовой среде, дают точное представление о ее поведении при эксплуатации. Вы должны стремиться к раннему обнаружению проблем в среде тестирования.

6.4.3 Проблемы тестирования

Без интеграционных тестов вы не узнаете, успешно ли обновляется серверный модуль или запись DNS, пока не проверите их вручную. Они ускоряют процесс проверки работоспособности вашего сценария IaC. Однако при интеграционном тестировании вы столкнетесь с некоторыми проблемами.

У вас могут возникнуть трудности с определением параметров конфигурации для тестирования. Стоит ли вам писать интеграционные тесты, чтобы убедиться, что *каждый* параметр конфигурации, настроенный вами в IaC, соответствует реальному ресурсу? Не обязательно!

Большинство инструментов уже имеют *приемочные тесты* (assertance test), которые создают ресурс, обновляют его конфигурацию и удаляют ресурс. Приемочные тесты подтверждают, что инструмент может выпускать новые изменения кода. Эти тесты необходимо проводить, чтобы убедиться в том, что инструмент способен поддерживать изменения в инфраструктуре.

Незачем тратить дополнительное время или усилия на написание специальных приемочных тестов. Ваши интеграционные тесты должны проверять, имеет ли *комбинация ресурсов* правильную конфигурацию и зависимости. Если вы пишете собственный сценарий автоматизации, вам нужно будет разработать интеграционные тесты для создания, обновления и удаления ресурсов.

Другая проблема связана с принятием решения о том, следует создавать или удалять ресурсы во время каждого теста либо запускать постоянную среду тестирования. На рис. 6.12 показано дерево решений о том, создавать новую тестовую среду для интеграционного теста или использовать постоянную среду.

В целом если конфигурации или модули не имеют слишком много зависимостей, вы можете создавать их, проверять и удалять. Однако если для создания вашей конфигурации или модуля требуется время либо наличие множества других ресурсов, выгоднее использовать постоянную среду тестирования.

К некоторым модулям нельзя применять подход создания и удаления при интеграционном тестировании. Я рекомендую запускать интеграционные тесты для модулей нижнего уровня, таких как сети или DNS, и избегать удаления ресурсов. Для этих модулей, как правило, достаточно обновлений «на ходу» с минимальными финансовыми затратами. В моей практике часто бывает удобнее протестировать рабочее обновление, а не создавать и удалять ресурс.

Ресурсы, созданные в ходе интеграционного тестирования модулей среднего уровня, например оркестраторов рабочей нагрузки, могут быть постоянными или временными в зависимости от размера модуля и ресурса. Чем больше модуль, тем больше вероятность

того, что он будет постоянным или долговечным. Вы можете запускать интеграционные тесты для модулей верхнего уровня, таких как развертывание приложений или SaaS, и каждый раз создавать и удалять ресурсы.

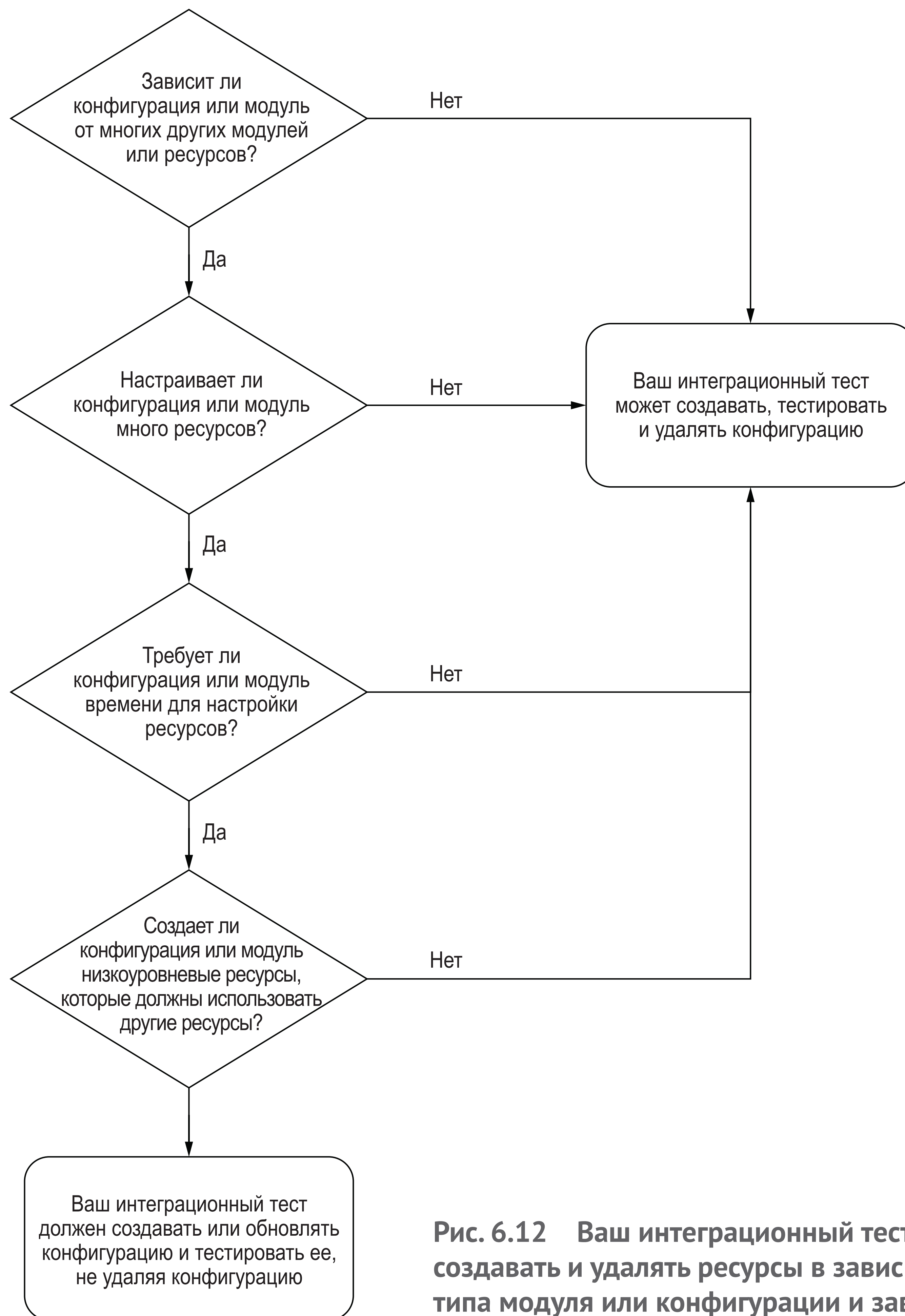


Рис. 6.12 Ваш интеграционный тест должен создавать и удалять ресурсы в зависимости от типа модуля или конфигурации и зависимостей

Постоянная тестовая среда имеет свои ограничения. Интеграционные тесты обычно занимают много времени, поскольку создание или обновление ресурсов требует времени. Поэтому старайтесь создавать модули меньшего размера, зависящие от меньшего количества ресурсов. Такая практика сокращает количество времени, необходимое для проведения интеграционного тестирования модуля.

Даже если вы предпочитаете небольшие конфигурации и модули с небольшим количеством ресурсов, интеграционные тесты часто становятся причиной увеличения расходов на оплату услуг провайдера инфраструктуры. Для ряда тестов требуются долгоживущие ресурсы, такие как сети, шлюзы и т. д. Сопоставьте стоимость проведения интеграционного теста с возможными затратами на неправильную настройку или поломку ресурса инфраструктуры.

Чтобы снизить стоимость запуска интеграционного или любого другого теста, можно воспользоваться *макетом* инфраструктуры. Некоторые платформы имитируют API провайдера инфраструктуры для локального тестирования. Я не рекомендую сильно полагаться на макеты. Провайдеры инфраструктуры часто меняют API, что влечет за собой сложные ошибки и непредсказуемое поведение системы, но эти ошибки не всегда отражаются в макетах. В главе 12 я обсуждаю методы ограничения затрат на среду тестирования без использования макетов.

6.5 Сквозные тесты

Хотя интеграционные тесты динамически анализируют конфигурацию и выявляют ошибки во время создания или обновления ресурса, они не указывают, можно ли *использовать* ресурс инфраструктуры по назначению.

Допустим, мы применяем модуль для создания приложения, называемого *службой* (service), в GCP Cloud Run. GCP Cloud Run развертывает службу в контейнере и возвращает конечную точку URL. Интеграционные тесты пройдены, следовательно, наш модуль правильно создает ресурс службы и разрешения на доступ к службе.

Как узнать, может ли кто-то получить доступ к URL приложения? На рис. 6.13 показано, как проверить работу конечной точки службы. Сначала мы создаем тест, который получает URL приложения в качестве выходных данных конфигурации вашей инфраструктуры. Затем мы выполняем HTTP-запрос к URL. Общее время выполнения занимает несколько минут, большая часть времени приходится на создание службы.

Мы создали тест для динамического анализа, который отличается от интеграционного теста и называется *сквозным тестом*. Он проверяет функциональность инфраструктуры с точки зрения конечного пользователя.

ОПРЕДЕЛЕНИЕ Сквозные тесты (end-to-end test) динамически анализируют ресурсы инфраструктуры и сквозные функциональные возможности системы, чтобы убедиться, что изменения IaC влияют на них, как ожидалось.

В нашем примере сквозной тест проверяет сквозной рабочий процесс конечного пользователя, обращающегося к странице. Он *не* проверяет успешную настройку инфраструктуры.

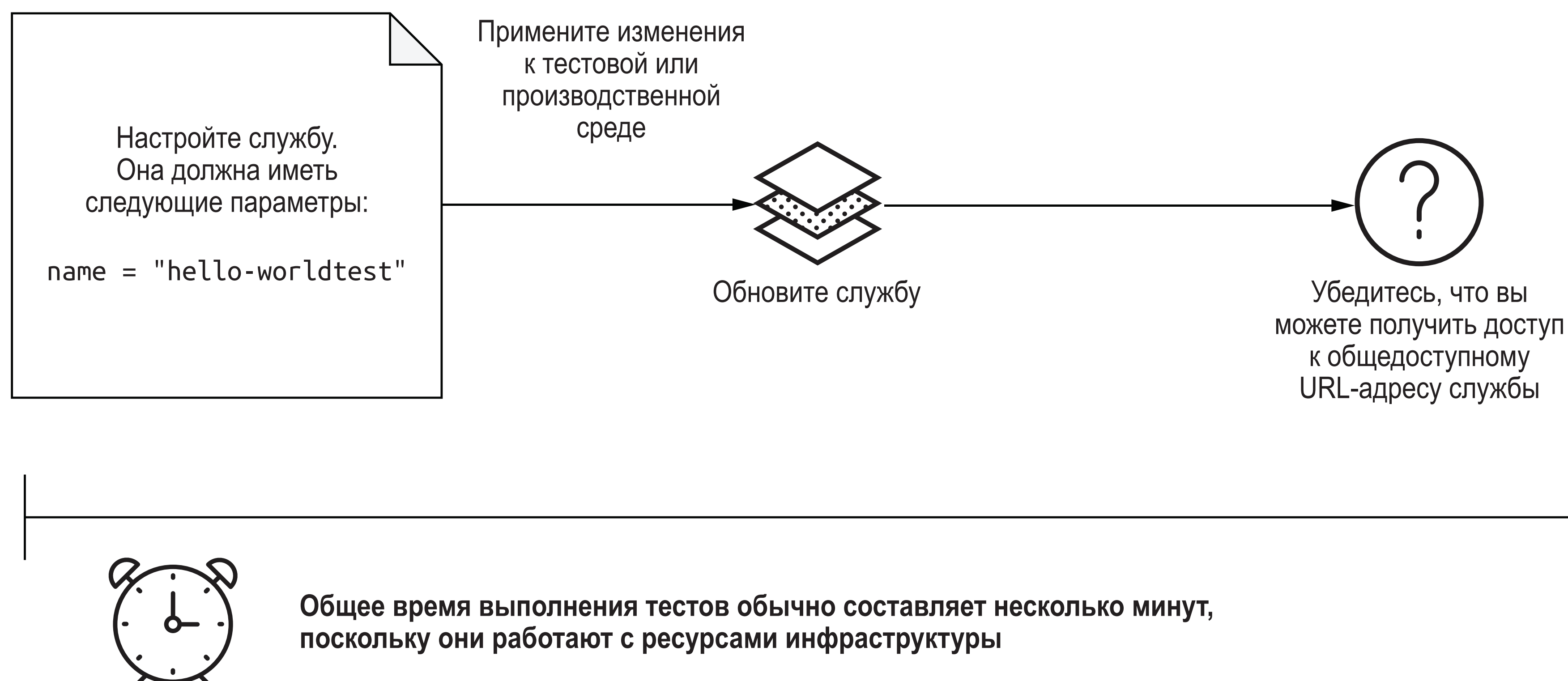


Рис. 6.13 Сквозные тесты проверяют рабочий процесс конечного пользователя путем доступа к веб-странице по адресу приложения

Сквозные тесты принципиально важны для проверки того, что ваши обновления не нарушают функциональность первоначального кода. Например, вы можете случайно обновить конфигурацию, которая позволяет прошедшим проверку подлинности пользователям получать доступ к адресу службы GCP Cloud Run. После применения изменения ваш сквозной тест завершается неудачей. Это означает, что некоторые (или все) пользователи не смогут получить доступ к службе.

Пример сквозного теста URL приложения на Python показан в листинге 6.5. Тест из этого примера должен выполнить запрос API к общедоступному адресу службы. Он использует тестовый стенд pytest для создания службы GCP Cloud Run, проверки URL-адреса работающей страницы и удаления службы из тестовой среды.

Листинг 6.5 Сквозной тест для службы GCP Cloud Run

```

from main import generate_json, SERVICE_CONFIGURATION_FILE
import os
import pytest
import requests
import test_utils

TEST_SERVICE_NAME = 'hello-world-test'

@pytest.fixture(scope='session')
def apply_changes():
    generate_json(TEST_SERVICE_NAME)
    assert os.path.exists(SERVICE_CONFIGURATION_FILE)
  
```

1

2

2

3

3


```

    assert test_utils.initialize() == 0
    yield test_utils.apply()
    assert test_utils.destroy() == 0
    os.remove(SERVICE_CONFIGURATION_FILE)

@pytest.fixture
def url():
    output, error = test_utils.output('url')
    assert error == b''
    service_url = output.decode(encoding='utf-8').split('\n')[0]
    return service_url

def test_url_for_service_returns_running_page(apply_changes, url):
    response = requests.get(url)
    assert "It's running!" in response.text

```

- ❶ Используя Terraform, инициализируем и развертываем службу с помощью файла Terraform JSON.
- ❷ Во время сеанса тестирования используем тестовый стенд pytest для применения конфигурации и создания тестовой службы в GCP.
- ❸ Создаем файл Terraform JSON, который использует модуль GCP Cloud Run.
- ❹ Удаляем службу GCP Cloud Run в тестовой среде, поэтому в нашем проекте GCP не будет постоянной службы.
- ❺ Используем тестовый стенд pytest для анализа выходных данных конфигурации URL-адреса службы.
- ❻ В тесте выполняется запрос API к адресу службы с использованием библиотеки requests Python.
- ❼ В ходе теста проверяется ответ службы, содержащий определенную строку, указывающую, что служба работает.

Эквиваленты AWS и Azure

AWS Fargate с Amazon Elastic Kubernetes Service (EKS) или Azure Container Instances (ACI) примерно соответствует GCP Cloud Run.

Стоит отметить, что если вы хотите запустить сквозной тест в промышленной среде, вам не нужно удалять службу. Обычно мы запускаем сквозные тесты в существующих средах, не создавая новые или тестовые ресурсы. Мы применяем изменения к существующей системе и запускаем тесты работающих ресурсов инфраструктуры.

Smoke-тест

Являясь разновидностью сквозного теста, smoke-тест¹ обеспечивает быстрое уведомление о том, нарушило ли изменение критически важ-

¹ Проверка «на дымок» – грубая проверка работоспособности простым включением устройства или запуском системы. Термин позаимствован из жаргона радиоинженеров. – *Прим. перев.*

ные бизнес-функции. Выполнение всех сквозных тестов может занять время, но вам необходимо как можно скорее устранить неполадки после обновления.

Smoke-тест позволяет убедиться, что изменение не привело к катастрофическим последствиям, и продолжить дальнейшее тестирование. Как однажды сказал мне аналитик по обеспечению качества: «Если вы включили прибор и из него пошел дым, то сразу ясно, что он неисправен. Нет смысла продолжать проверку».

Более сложные системы инфраструктуры выигрывают от сквозных тестов (end-to-end), поскольку они становятся основным показателем того, повлияли ли изменения на критически важную бизнес-функциональность. В результате такие тесты помогают проверять конфигурации, использующие паттерны «компоновщик» или «одиночка». Обычно сквозные тесты не запускаются для модулей, если только они не содержат множество ресурсов и зависимостей.

Я применяю большую часть своих сквозных тестов к сетевым или вычислительным ресурсам. Например, можно написать несколько тестов для проверки пиринга между своими сетями. Тесты выделяют сервер в каждой сети и проверяют, могут ли серверы подключаться друг к другу.

Другой вариант использования сквозных тестов предполагает отправку задания оркестратору рабочей нагрузки и его выполнение. Этот тест определяет, правильно ли работает оркестратор рабочей нагрузки при развертывании приложений. Однажды мне довелось выполнять сквозные тесты, которые выдавали запросы протокола передачи гипертекста (HTTP) с различной полезной нагрузкой, чтобы убедиться, что вышестоящие службы могут обращаться друг к другу без сбоев, независимо от размера полезной нагрузки или протокола.

За рамками сетевых или вычислительных сценариев сквозные тесты могут проверить ожидаемое поведение любой системы. Если вы используете управление конфигурацией с помощью инструмента инициализации инфраструктуры, ваши сквозные тесты проверяют, можете ли вы подключиться к серверу и запустить ожидаемые функции. Для проверки мониторинга и оповещений вы можете запускать сквозные тесты, имитирующие ожидаемое поведение системы, отслеживать сбор показателей и проверять срабатывание оповещения.

Однако сквозные тесты являются наиболее затратными с точки зрения времени и ресурсов. Для большинства сквозных тестов нужны все доступные ресурсы инфраструктуры. В результате мы можем запускать полноценные сквозные тесты только для производственной инфраструктуры. Мы не можем запускать их в тестовой среде, поскольку закупка достаточного количества ресурсов для теста часто обходится слишком дорого.

6.6 Другие тесты

Помимо модульных, контрактных, интеграционных и сквозных, существуют и другие типы тестов. Например, предположим, что вы хотите внедрить изменение конфигурации на рабочем сервере, которое уменьшает объем памяти. Однако вы не знаете, повлияет ли сокращение памяти на всю систему.

На рис. 6.14 показано, что вы можете проверить, не влияет ли ваше изменение на систему, с помощью мониторинга системы. Мониторинг постоянно агрегирует показатели заполнения памяти сервера. Если вы получаете предупреждение о том, что заполнение памяти сервера достигает определенного процента от общей емкости, это означает, что обновление может негативно повлиять на всю систему.

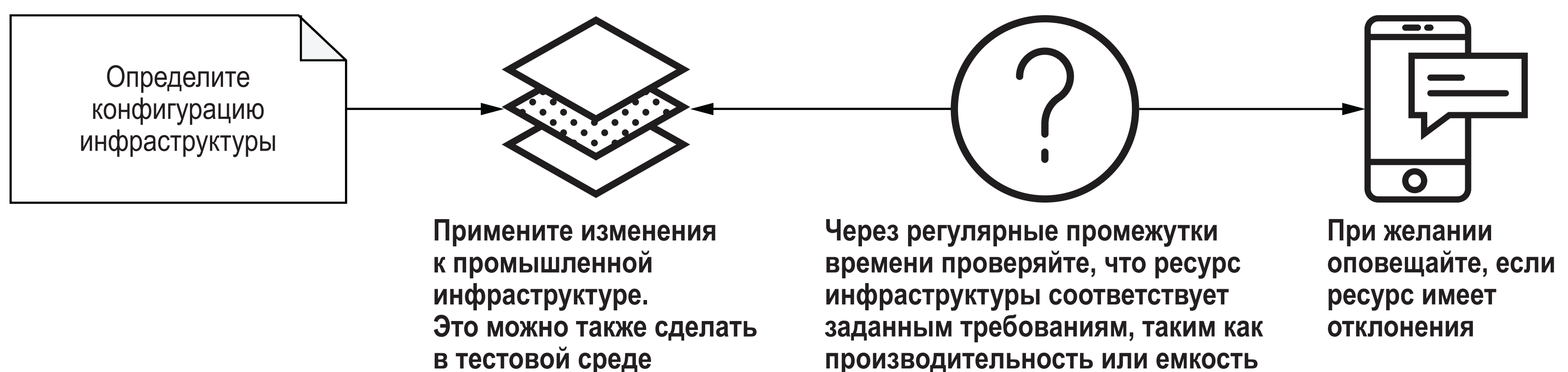


Рис. 6.14 Непрерывные тесты выполняются через короткие промежутки времени, чтобы убедиться, что набор метрик не превышает порогового значения

Мониторинг представляет собой непрерывное тестирование через регулярные и частые интервалы времени для проверки того, что показатели не выходят за рамки пороговых значений.

ОПРЕДЕЛЕНИЕ *Непрерывные тесты* (например, мониторинг) выполняются через регулярные и частые интервалы времени для проверки соответствия текущего значения ожидаемому значению или диапазону значений.

Непрерывное тестирование включает в себя мониторинг системных показателей и событий безопасности (например, когда пользователь root входит в систему на сервере). По сути, это непрерывный динамический анализ в среде активной инфраструктуры. Большинство непрерывных тестов имеют форму предупреждений, которые уведомляют вас о возникших проблемах.

Вы можете столкнуться также с *регрессионным тестом*. Например, вы можете выполнять тест в течение определенного периода времени, чтобы проверить, соответствует ли конфигурация вашего сервера ожиданиям вашей организации. Регрессионные тесты проводятся регулярно, но не требуют регулярного мониторинга или

других форм непрерывного тестирования. Вы можете запускать их каждые несколько недель или месяцев, чтобы выявлять наличие нестандартных ручных изменений.

ОПРЕДЕЛЕНИЕ *Регрессионные тесты* (regression test) запускаются периодически в течение длительного периода времени, чтобы проверить, соответствует ли конфигурация инфраструктуры ожидаемому состоянию или функциональности. Они помогают смягчить последствия от дрейфа конфигурации.

Для запуска непрерывных и регрессионных тестов часто требуется специальное программное обеспечение или системы. Регулярное тестирование гарантирует, что производственная инфраструктура будет работать с ожидаемой функциональностью и производительностью. Эти тесты также лежат в основе автоматических систем реагирования на аномалии.

Например, системы, настроенные с применением IaC и непрерывных тестов, могут использовать автоматическое масштабирование для настройки ресурсов на основе таких показателей, как ЦП или память. Эти системы также могут реализовывать другие механизмы самовосстановления, такие как перенаправление трафика на более старую версию приложения в случае ошибок.

6.7 Выбор тестов

Я рассказала о наиболее распространенных тестах инфраструктуры, от модульных до сквозных. Однако нужно ли разрабатывать их все? На что вам следует потратить свое время и силы при их подготовке? Ваша стратегия тестирования инфраструктуры будет меняться в зависимости от сложности и скорости развития системы. Поэтому стандартного решения не существует. В результате вы будете постоянно оценивать, какие тесты помогут вам обнаружить проблемы с конфигурацией до вывода в промышленную эксплуатацию.

В качестве ориентира для выбора стратегии тестирования инфраструктуры я использую пирамидальную диаграмму. На рис. 6.15 самая широкая часть пирамиды указывает на то, что вам следует отдавать предпочтение этому типу теста, а самая узкая часть указывает на то, что соответствующие тесты нужно проводить реже. На вершине пирамиды находятся сквозные тесты, которые могут стоить больше времени и денег, поскольку применяются к системам производственной инфраструктуры. Внизу пирамиды находятся модульные тесты, которые выполняются за секунды и не нуждаются в полной инфраструктуре.

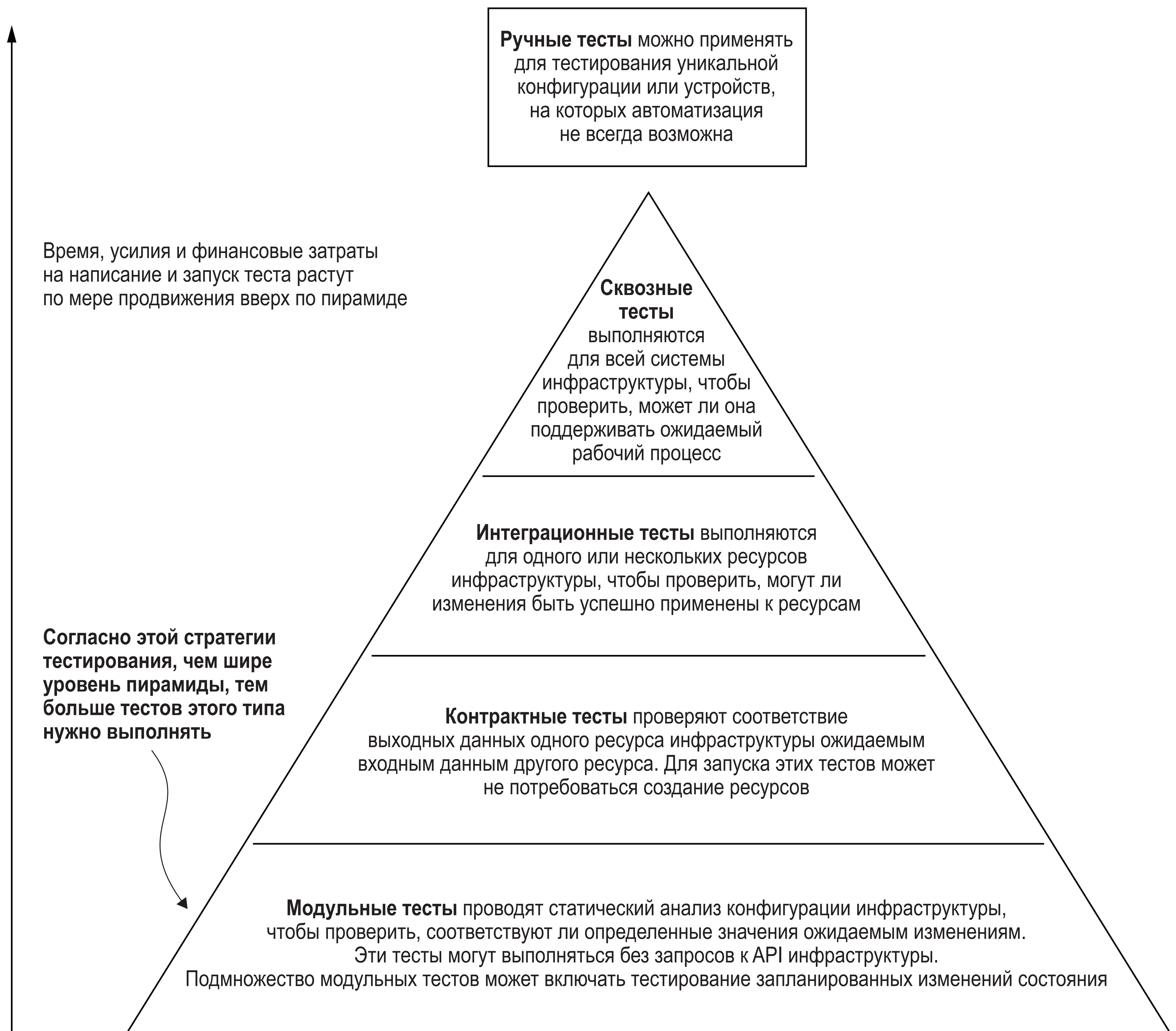


Рис. 6.15 Судя по пирамиде тестирования, модульных тестов должно быть больше, чем сквозных, поскольку их выполнение требует меньше времени, денег и ресурсов

Эта стратегия, называемая *пирамидой тестирования*, служит основой для выбора различных типов тестов, их объема и частоты. Я позаимствовала эту пирамиду тестирования из традиционной разработки программного обеспечения и адаптировала к инфраструктуре, модифицировав ее с учетом инструментов и ограничений инфраструктуры.

ОПРЕДЕЛЕНИЕ *Пирамида тестирования* (test pyramid) служит ориентиром для вашей общей стратегии тестирования. По мере подъема по пирамиде тот или иной тип теста будет стоить больше времени и денег.

В действительности ваша пирамида тестирования может иметь форму прямоугольника или груши, иногда с отсутствующими уровнями. Вы не будете и не должны писать каждый тип теста для каж-

дой конфигурации инфраструктуры. Когда тестов слишком много, их становится непрактично поддерживать.

В некоторых случаях соблюдение идеальной пирамиды тестирования может оказаться непрактичным. Однако избегайте того, что я в шутку называю *дорожным указателем* (signpost)¹ тестировщика. Такой указатель отражает лишь беспорядочное множество ручных тестов и ничего больше.

6.7.1 Стратегия модульного тестирования

Я упоминала о практике тестирования модулей перед их выпуском в главе 5. Давайте вернемся к тому примеру, где мы обновили модуль базы данных до PostgreSQL 12. Вместо того чтобы вручную создавать модуль и проверять, работает ли он, мы добавляем ряд автоматизированных тестов. Они проверяют форматирование модуля и создают базу данных в изолированной среде модульного тестирования.

На рис. 6.16 показан обновленный рабочий процесс выпуска модуля с помощью модульных, контрактных и интеграционных тестов, которые вы можете добавить, чтобы проверить работу вашего модуля. После прохождения контрактных тестов вы запускаете интеграционный тест, который настраивает модуль базы данных в сети и проверяет, работает ли база данных. После завершения интеграционного теста вы удаляете тестовую базу данных, созданную модулем, и выпускаете модуль.

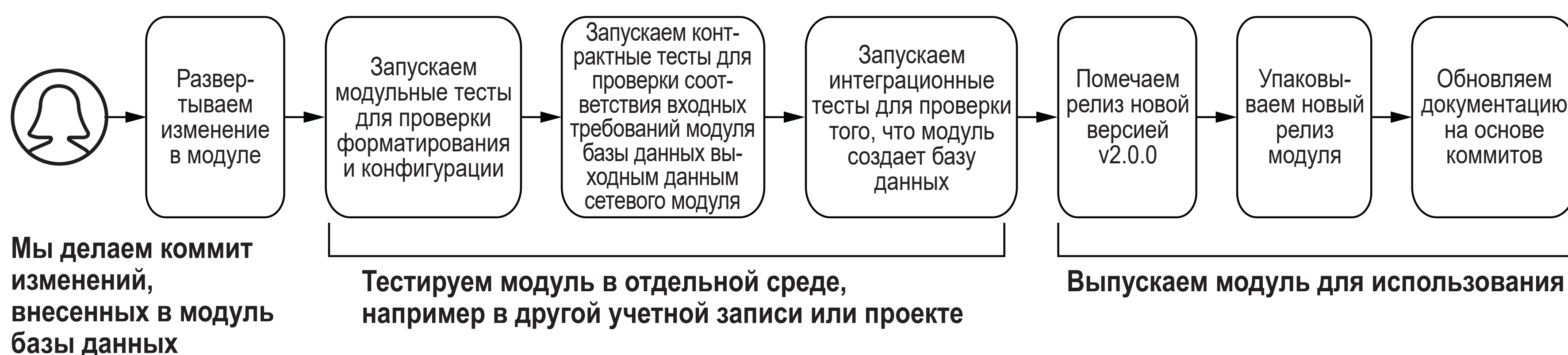


Рис. 6.16 Мы можем разбить этап тестирования рабочего процесса выпуска модуля, включив в него модульные, контрактные и интеграционные тесты

Сочетание модульных, контрактных и интеграционных тестов адекватно показывает, будет ли модуль работать правильно. Модульные тесты проверяют форматирование модуля и стандартные конфигурации вашей команды. Мы запускаем их в первую очередь и моментально получаем уведомление о любых ошибках форматирования или конфигурации.

¹ Автор имеет в виду придорожный столб, перегруженный бессистемной кучей стрелок-указателей, показывающих в разные стороны. – Прим. перев.

Затем мы запускаем несколько контрактных тестов. В случае модуля базы данных мы проверяем, соответствует ли входной идентификатор сети модуля базы данных выходному идентификатору сети сетевого модуля. Обнаружение этих ошибок позволит выявить проблемы между зависимостями на более ранних этапах процесса развертывания.

Если вам нужно обеспечить правильную конфигурацию, правильную логику модуля и соответствие между входными и выходными данными, сосредоточьтесь на модульном или контрактном тестировании. Рабочий процесс тестирования, показанный на рис. 6.16, лучше всего подходит для модулей, использующих паттерны фабричный, «строитель» или «прототип». Эти паттерны изолируют наименьшее подмножество компонентов инфраструктуры и предоставляют гибкий набор переменных для настройки вашими товарищами по команде.

В зависимости от стоимости вашей среды разработки вы можете написать несколько интеграционных тестов для работы с временными ресурсами инфраструктуры, которые вы удалите в конце теста. Потратив некоторое время и усилия на написание тестов для модулей со множеством входов и выходов, вы будете уверены, что изменения не повлияют на остальную конфигурацию и что сам по себе модуль способен успешно работать.

Упражнение 6.1

Допустим, вы заметили, что новая версия модуля балансировки нагрузки нарушает вашу конфигурацию DNS. Коллега по команде обновил модуль, чтобы он выдавал частные IP-адреса вместо общедоступных. Что вы можете сделать, чтобы команда твердо усвоила, что модулю нужны общедоступные IP-адреса:

- a** создать отдельный модуль балансировки нагрузки для частных IP-адресов;
- b** добавить контрактные тесты модуля, чтобы убедиться, что модуль выводит как частные, так и общедоступные IP-адреса;
- c** обновить документацию модуля, указав, что ему необходимы общедоступные IP-адреса;
- d** запустить интеграционные тесты модуля и убедиться, что IP-адрес общедоступен.

Ответы на упражнения вы найдете в приложении В.

6.7.2 Стратегия тестирования конфигурации

Конфигурации инфраструктуры для активных сред используют более сложные паттерны, такие как «одиночка» или «компоновщик». Конфигурация, основанная на паттернах «одиночка» или «компоновщик», имеет множество зависимостей от инфраструктуры и часто ссылается на другие модули. Добавление сквозных тестов (end-to-

end) в рабочий процесс тестирования может помочь выявить проблемы между инфраструктурой и модулями. Представьте, что у нас есть одноэлементная конфигурация с сервером приложений в сети.

На рис. 6.17 показаны шаги, которые нужно выполнить после обновления размера сервера. После отправки изменения в систему контроля версий мы разворачиваем его в тестовой среде. Наш рабочий процесс тестирования начинается с модульных тестов, позволяющих быстро проверить форматирование и конфигурацию.

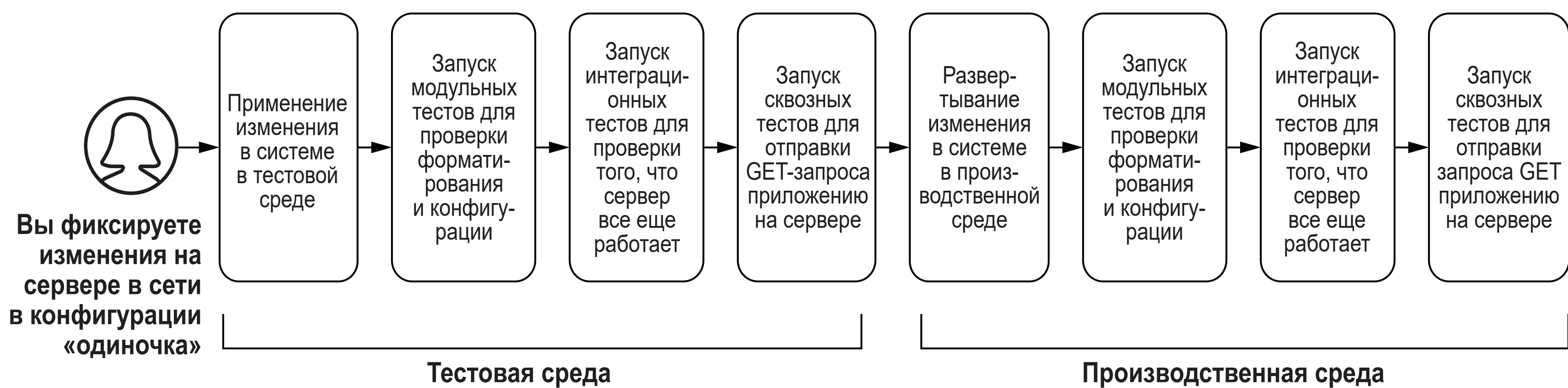


Рис. 6.17. IaC (инфраструктура как код), использующая паттерны «одиночка» и «компоновщик», должна выполнять модульные, интеграционные и сквозные тесты (end-to-end) в тестовой среде перед инициализацией изменений в промышленной среде

Затем мы запускаем интеграционные тесты, чтобы применить изменения и убедиться, что сервер по-прежнему работает и имеет новый размер. Мы завершаем проверку сквозным тестом для проверки всей системы. Сквозной тест отправляет запрос HTTP GET конечной точке приложения. На рис. 6.17 показано повторение процесса, чтобы убедиться, что система сохранит работоспособность. Тот факт, что вы успешно создали или обновили сервер, не означает, что размещенное на нем приложение может обслуживать запросы! В случае сложной инфраструктуры вам потребуются дополнительные тесты для проверки зависимостей или связей между элементами инфраструктуры. Сквозное тестирование помогает сохранить функциональность системы.

Повторение одних и тех же тестов в тестовой и промышленной средах лежит в основе контроля качества. Если существует дрейф конфигурации между тестовой и промышленной средами, наши тесты смогут его обнаружить. У нас есть возможность включать и отключать определенные тесты в зависимости от среды.

Построение образа и управление конфигурацией

Тесты для инструментов создания образов и управления конфигурацией используют аналогичный подход к тестированию конфигурации для инструментов инициализации инфраструктуры. Создание образа модульного тестирования или метаданных управления конфигурацией предусматривает проверку конфигурации. Вам не нужны контракт-

ные тесты, если вы не меняете разбиение конфигурации на модули, но выполняете тестирование модулей. Интеграционные тесты следует запускать в тестовой среде, чтобы проверить, что сервер успешно запускается с новым образом или применяет правильную конфигурацию. Сквозные тесты гарантируют, что ваши новые образы и конфигурации не повлияют на функциональность системы.

Упражнение 6.2

Допустим, вы добавляете правила брандмауэра, чтобы разрешить приложению доступ к новой очереди. Какая комбинация тестов с точки зрения вашей команды будет наиболее полезной для внесения изменений:

- a модульные и интеграционные тесты;
- b контрактные и сквозные тесты;
- c контрактные и интеграционные тесты;
- d модульные и сквозные тесты.

Ответы на упражнения вы найдете в приложении В.

6.7.3 Как составить набор наиболее полезных тестов

На начальном этапе разработки полезных тестов хорошо помогают стандартные стратегии тестирования модулей и конфигураций. На рис. 6.18 показаны типы тестов, которые вы можете применить для проверки модулей и конфигураций. С модулями связаны модульные, контрактные и интеграционные тесты, а для конфигураций предназначены модульные, интеграционные и сквозные тесты.

Как узнать, *когда* нужно разрабатывать тест? Представьте, что вашему коллеге по команде откуда-то известно, что пароль базы данных должен состоять из букв и цифр с ограничением в 16 символов. Однако вы можете не знать об этом факте, пока не обновите модуль для использования 24-символьного пароля, не развернете изменение и не обнаружите спустя пять минут, что обновление завершилось неудачно.

Я рассматриваю практику обновления тестов как способ превращения «неизвестных известных» в «известные известные» в вашей системе. Фактически мы используем наблюдаемость системы для обнаружения «неизвестных неизвестных» и мониторинг для отслеживания «известных неизвестных». На рис. 6.19 показано превращение разрозненных знаний (неизвестные известные), которые знает кто-то другой, в тесты (известные известные) на основе командных знаний. Новые тесты часто отражают разрозненные знания, которые команда должна знать и учитывать.

Хороший тест передает знания остальной команде. Не всегда нужно создавать новый тест. Иногда удастся найти существующий под-

ходящий тест. Используйте тест, чтобы ваша команда не повторяла проблемы.

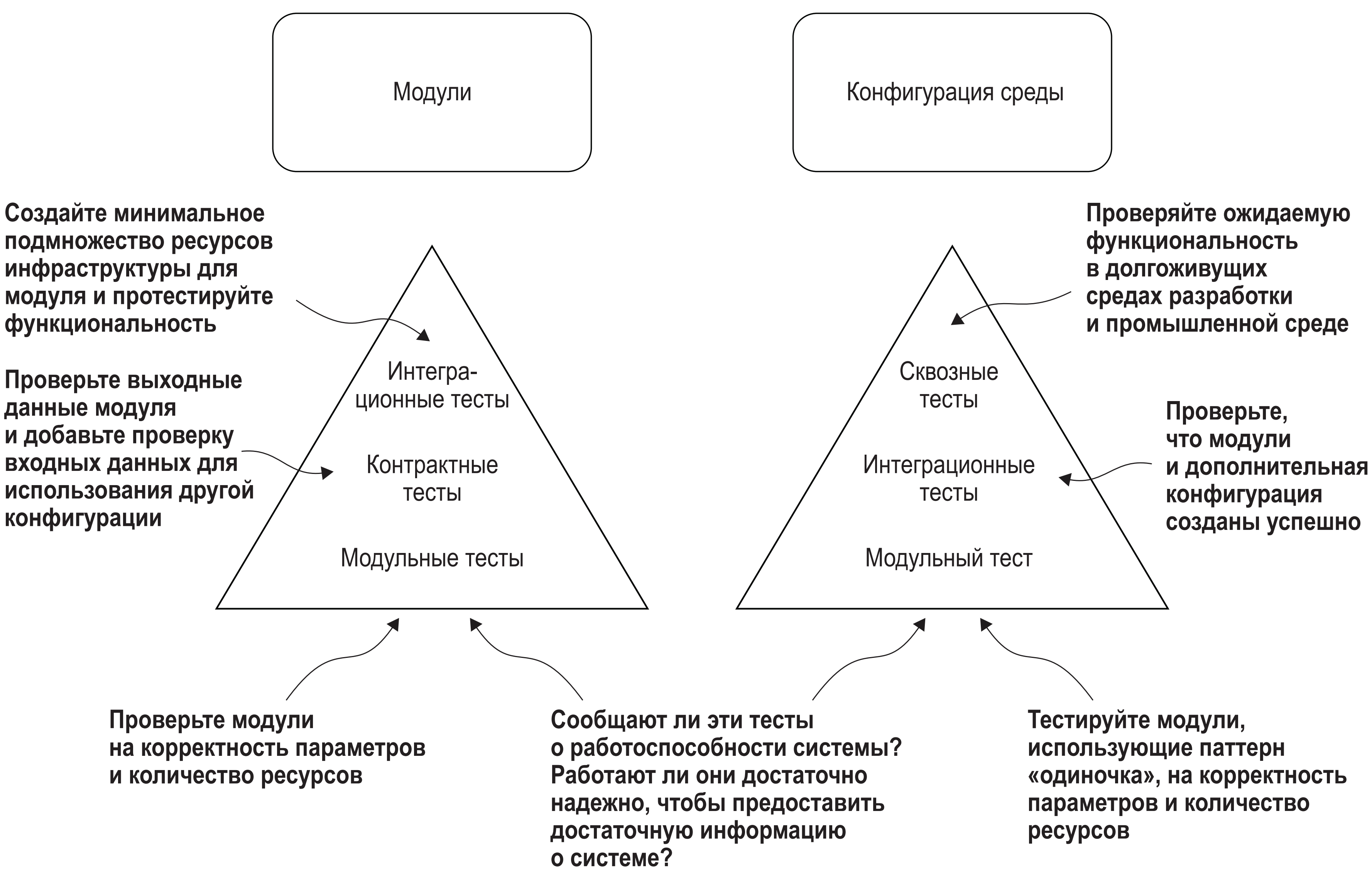


Рис. 6.18 Ваш подход к тестированию будет различаться в зависимости от того, пишете вы конфигурацию модуля или среды

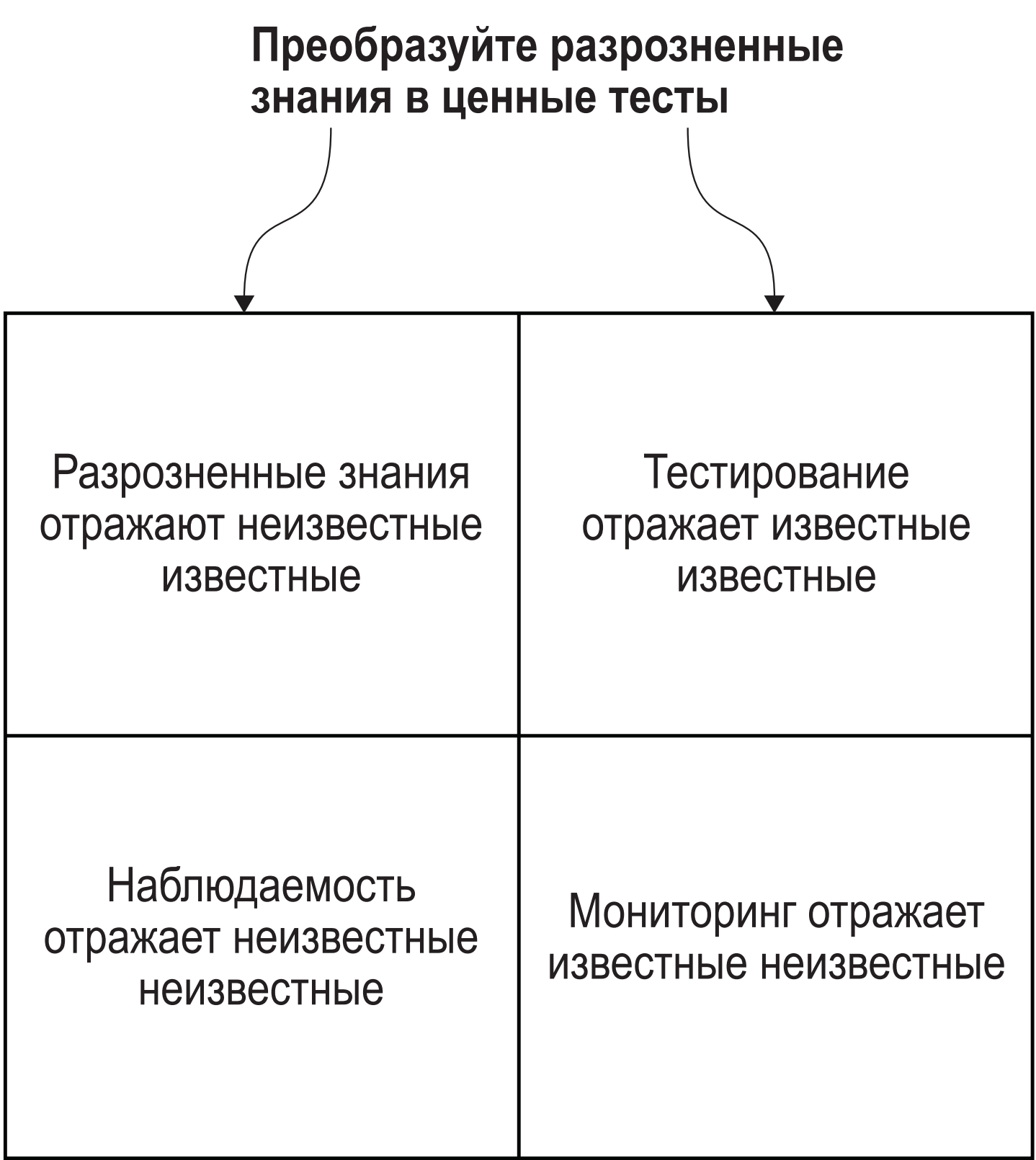


Рис. 6.19 Тестирование инфраструктуры преобразует разрозненные знания, которыми может обладать кто-то другой, в тест, отражающий знания команды

Помимо добавления тестов, иногда приходится удалять тесты. Тесты тоже не застрахованы от ошибок. Вы можете написать тест и обнаружить, что в половине случаев он терпит неудачу. Он не предоставляет полезной информации и не повышает вашей уверенности в системе из-за собственной ненадежности. Удаление теста

очищает ваш набор тестов и помогает избавиться от нестабильных тестов, которые постоянно терпят неудачу, но не указывают на настоящий сбой в системе.

Кроме того, вы будете удалять тесты, потому что они вам больше не нужны. Например, вам могут не потребоваться контрактные тесты для каждого модуля или интеграционные тесты для каждой конфигурации среды. Всегда спрашивайте себя, имеют ли тесты ценность и достаточно ли они надежны, чтобы вы могли получить достаточную информацию о системе.

В следующей главе показано, как добавить тесты в пайплайн поставки IaC. Даже если вы не планируете автоматизировать рабочий процесс тестирования, у вас есть возможность изучить, как изменения способны повлиять на вашу инфраструктуру.

Краткое содержание главы

- Стратегию тестирования можно представить в виде пирамиды. Чем выше расположен тест в пирамиде, тем дороже он обходится.
- Модульные тесты проверяют статические параметры в модулях или конфигурациях.
- Контрактные тесты проверяют, что входные и выходные данные модуля соответствуют ожидаемым значениям и форматам.
- Интеграционные тесты создают тестовые ресурсы, проверяют их конфигурацию и существование и удаляют их.
- Сквозные тесты проверяют, может ли конечный пользователь инфраструктурной системы воспользоваться ожидаемыми функциями.
- Для проверки модулей, использующих паттерны фабричный, «строитель» или «прототип», лучше подходят модульные, контрактные и интеграционные тесты.
- Конфигурации, использующие паттерны «компоновщик» или «одиночка», примененные к средам, выигрывают от модульных, интеграционных и сквозных тестов (end-to-end).
- Примерами других тестов могут служить непрерывный мониторинг системных показателей, регрессионные тесты внеплановых ручных изменений или тесты безопасности на предмет неправильных конфигураций.

7 Модели непрерывной поставки и ветвления

Основные темы главы:

- построение конвейеров поставки, позволяющих избежать сбоев в промышленных средах;
- выбор модели ветвления Git;
- управление изменениями ресурсов инфраструктуры внутри вашей команды.

В предыдущих главах вы научились разрабатывать шаблоны для модулей и зависимостей. Вы также освоили некоторые общие методы разработки инфраструктуры в виде кода и модулей совместного использования. Паттерны, методы и рабочие процессы связаны с выполнением множества разнообразных действий.

Более того, многие рабочие процессы требуют тщательной координации изменений. Однажды вы попытаетесь внести изменения и обнаружите, что обновления вашего коллеги по команде перезаписывают ваши обновления! Как организовать разрешение конфликтов в процессе разработки?

Одно из решений заключается в отправке запросов на изменение в систему обработки заявок. Например, если вы хотите изменить

конфигурацию сервера, вам необходимо заполнить запрос на изменение в вашей системе обработки заявок. Затем этот запрос рассматривается вашими коллегами (обычно вашей командой) и ответственным за изменения на уровне компании.

Большинство компаний используют этот процесс, называемый *управлением изменениями*, чтобы гарантировать отсутствие конфликтов изменений. Управление изменениями инфраструктуры предусматривает подробное описание этапов развертывания и возможного отката предстоящих изменений в случае возникновения нештатной ситуации.

ОПРЕДЕЛЕНИЕ *Управление изменениями* (change management) в инфраструктуре – это процесс, который способствует безопасному внесению изменений в инфраструктуру. Часто он включает в себя детализацию и анализ изменений экспертами компании перед их утверждением.

Управление изменениями опирается на экспертную оценку, позволяющую предотвратить перезапись изменений, сделанных другими членами команды. В данном примере централизованное согласование с ответственными на уровне компании образует систему *входного контроля качества*. Она следит за тем, чтобы запросы на изменение не ставили под угрозу безопасность или доступность системы. Только после того, как изменение прошло проверку, вы можете запланировать его применение и изменить конфигурацию сервера.

ОПРЕДЕЛЕНИЕ Система *входного контроля качества* (quality gates) для IaC обеспечивает безопасность, доступность и отказоустойчивость системы посредством проверки описания инфраструктуры или тестирования его работы.

Как управление изменениями и контроль качества помогают решать конфликты? Представьте, что вы применили данный процесс в случае, когда вы и ваш товарищ по команде пытаетесь изменить конфигурацию серверов. На рис. 7.1 показано, что вы и ваш коллега отправляете запросы на изменения в систему обработки заявок. Остальные коллеги, работающие в этой же организации, вручную просматривают каждую заявку на изменение и определяют, что изменение вашего товарища по команде оказывает минимальное влияние на пользователей. Они переносят работы по вашей заявке, чтобы не возникали конфликты.

Управление изменениями может занять несколько недель. Ручная проверка не позволяет выявить все проблемы и не предотвращает все конфликты при изменении инфраструктуры. Вам еще нужно понять, как отменить изменения. Вы узнаете об исправлении неудачных изменений в главе 11.



Рис. 7.1 Совместная работа в рамках IaC предполагает оптимизацию процесса коллегиального и организационного анализа изменений

Вместо того чтобы полагаться на ручное управление изменениями, можно воспользоваться преимуществами IaC для передачи изменений через код и автоматизации управления изменениями. В этой главе основное внимание уделяется оптимизации управления изменениями путем масштабирования и автоматизации процессов разработки IaC в вашей команде и компании. В своей практике вы неизбежно столкнетесь с проблемой, когда люди работают над одними и теми же или зависимыми ресурсами, стараясь при этом не сломать промышленную систему.

Построение образа и управление конфигурацией

На протяжении всей этой книги я уделяю много внимания вариантам конфигурирования инфраструктуры. Способы создания образов и управления конфигурацией должны соответствовать общей схеме конвейеров поставки, описанной в этой главе. Паттерны и методы оценки изменений инфраструктуры и внедрения автоматического тестирования остаются одинаковыми для всех вариантов использования.

7.1 Поставка изменений в производство

Как мы управляем изменениями IaC в промышленной среде? Мы применяем стандартные методы разработки программного обеспечения, такие как *непрерывная интеграция, поставка или деплой* (continuous integration, delivery or deployment, CI/CD), для упорядочения

изменений кода от различных разработчиков и подготовки к выпуску IaC в промышленную среду.

Для автоматизации выпуска изменений и управления ими процессы CI/CD нуждаются в автоматическом тестировании. Я объясню, как можно автоматизировать изменения инфраструктуры и максимально эффективно использовать ее преимущества. Мы будем использовать методы тестирования, о которых вы узнали из предыдущей главы, с шаблонами конвейеров поставки, описанными в этой главе.

7.1.1 Непрерывная интеграция

Вспомните конфликт в IaC, с которым вы столкнулись вместе со своим товарищем по команде. Вы не знали, что обновление, выполненное вашим коллегой, повлияет на ваше обновление, и наоборот. Как автоматически выявлять конфликты до того, как коллеги столкнутся с вашими изменениями?

Одно из решений, представленных на рис. 7.2, заключается в том, чтобы попросить вашу команду регулярно вносить свои изменения в основную конфигурацию IaC. Если ваша команда постоянно *интегрирует* свои изменения в основную конфигурацию, вы и ваш товарищ по команде сможете выявить конфликты раньше, прежде чем они перезапишут ваши обновления.

Иными словами, вы должны применять непрерывную интеграцию (CI), чтобы объединять ваши изменения с основной конфигурацией несколько раз в день и проверять, не конфликтуют ли они с обновлениями других разработчиков.

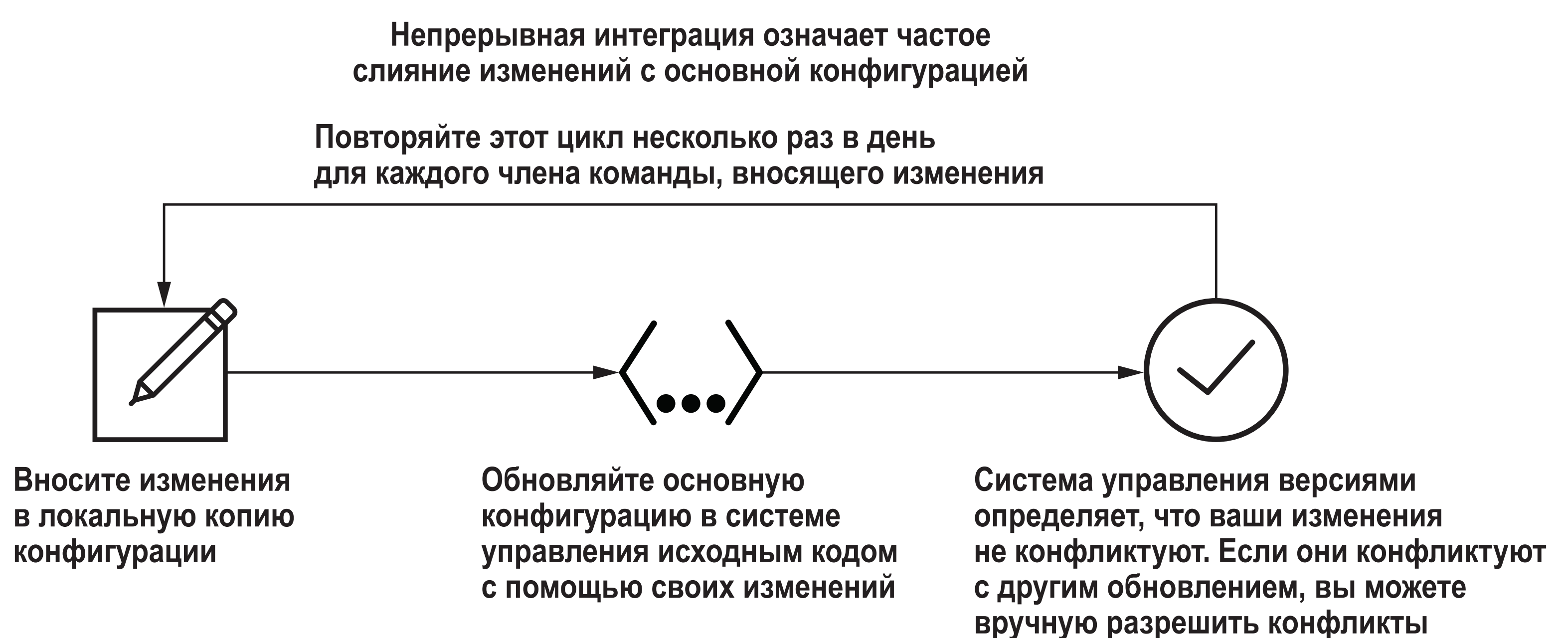


Рис. 7.2 Непрерывная интеграция предполагает частое слияние изменений с основной конфигурацией, что позволяет раньше выявлять конфликты между изменениями

Как часто следует выполнять *слияние* (merge)? Понимание того, когда следует выполнять слияние, требует некоторого опыта и зависит от типа изменения, которое вы хотите внести. Как правило, я выполняю

слияние, когда накапливаю несколько минимальных изменений конфигурации, которые (вероятно) не сломают систему. Иногда это означает, что слияние происходит несколько раз в день. В более сложных и потенциально опасных случаях я выполняю слияние один или два раза в день. Остальная часть команды продолжает работать над изменениями в своем темпе и делает слияния по необходимости.

Каждый раз, когда член команды хочет внести обновления, инструмент сборки должен запустить рабочий процесс проверки изменений и их развертывания. На рис. 7.3 показан пример рабочего процесса, запускаемого инструментом сборки. Этот процесс проверяет IaC на наличие конфликтов слияния, запускает модульные тесты для проверки форматирования и приостанавливает работу для согласования с другими членами команды. После прохождения коллегиальной проверки инструмент сборки развертывает изменения в рабочей среде.



Рис. 7.3 Фреймворк CI в конвейере поставки выполняет автоматизированные модульные тесты перед ожиданием ручного утверждения запуска в производство

Этот рабочий процесс можно рассматривать как часть *конвейера поставки*. Конвейер организует и автоматизирует этапы сборки, тестирования, развертывания и выпуска IaC.

ОПРЕДЕЛЕНИЕ *Конвейер поставки* (delivery pipeline) для IaC упорядочивает и автоматизирует этапы рабочего процесса по созданию, тестированию, развертыванию и применению изменений в инфраструктуре.

Конвейер поставки инфраструктуры начинается с выявления конфликтов конфигурации или синтаксических ошибок. Модульное тестирование в конвейере CI дает нам некоторую уверенность в отсутствии конфликтов изменений. Затем мы можем отправить изменение своей команде или компании на рассмотрение. Конвейер автоматически применяет его к инфраструктуре.

Почему вам следует разработать конвейер поставки и добавить его в свой инструмент сборки? Возможно, вы не помните все шаги,

необходимые для выпуска изменения в рабочую среду. Конвейер поставки кодифицирует процесс, поэтому вам не нужно его запоминать. Наличие конвейера поставки инфраструктуры позволяет масштабировать изменения инфраструктуры последовательно и воспроизводимо, независимо от ресурсов.

7.1.2 Непрерывная поставка

Допустим, вы использовали CI для объединения изменений и проверки на конфликты, но как узнать, работает ли система так, как вы ожидаете? CI проверяет форматирование и стандарты, но до применения вы не узнаете, работает ли конфигурация, как вы хотели. Модульный тест вселяет в вас некоторую надежду, но для окончательной уверенности нужны дополнительные тесты.

На рис. 7.4 показан доработанный рабочий процесс CI для конфигурации IaC. Мы расширили конвейер поставки дополнительными этапами после модульных тестов. Прежде чем отправить изменение на экспертную оценку, мы развертываем конфигурацию в тестовой среде и проверяем ее с помощью интеграционных и сквозных тестов. После экспертной оценки мы вносим изменения в рабочую среду и повторно запускаем сквозные тесты для проверки функциональности рабочей среды.

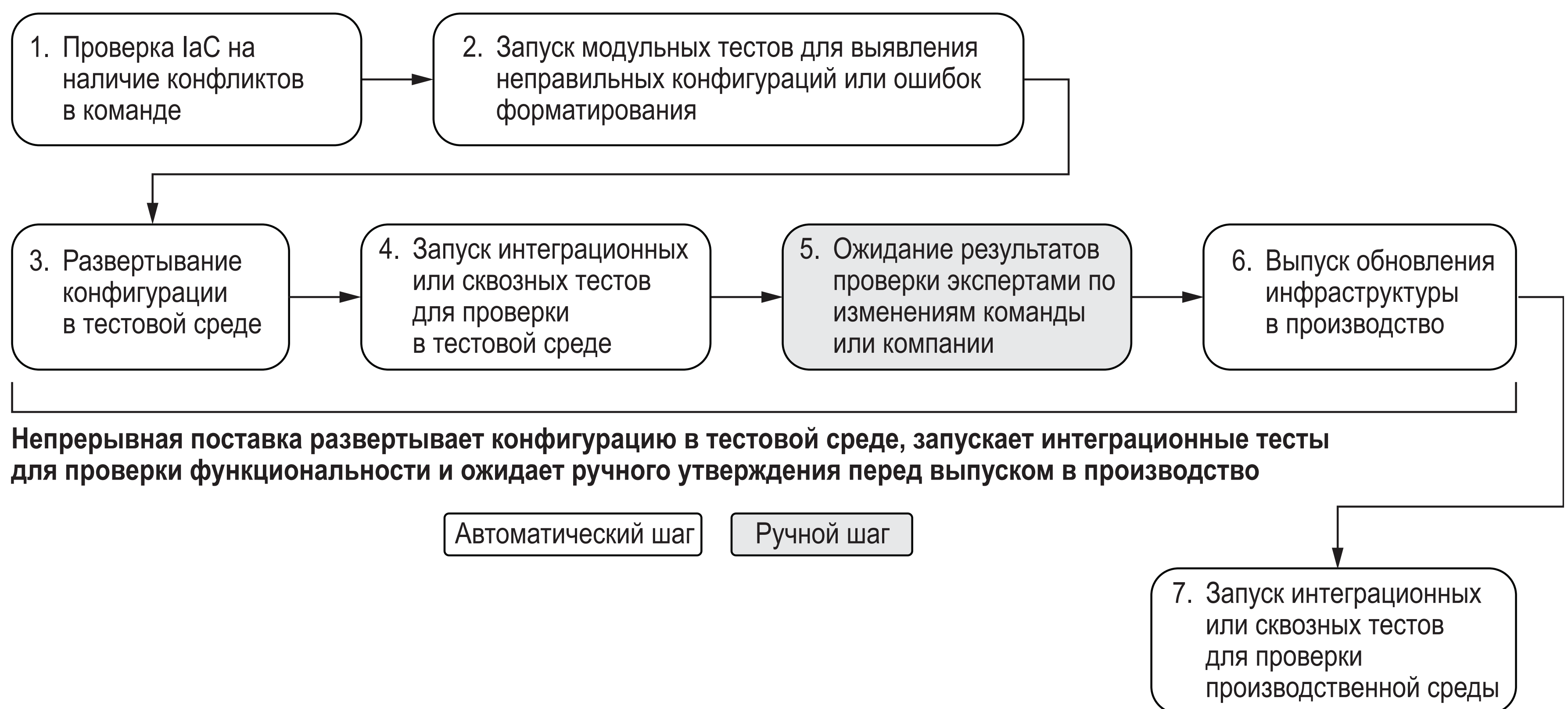


Рис. 7.4 Система непрерывной поставки автоматически развертывает и запускает изменения инфраструктуры в среде тестирования и ожидает ручного утверждения для выпуска в производство

Мы дополнили конвейер, применив технологию *непрерывной поставки*. В соответствии с подходом CD в конвейер поставки добавляют этап, который развертывает конфигурацию инфраструктуры в среде тестирования для интеграции или сквозного тестирования после прохождения модульных тестов.

ОПРЕДЕЛЕНИЕ *Непрерывная поставка* (continuous delivery, CD) для IaC развертывает изменение инфраструктуры в тестовой среде для интеграции или сквозного тестирования после слияния изменений в репозитории. В процесс CD может входить ручная проверка качества перед применением изменений в промышленной среде.

Всякий раз, когда кто-то отправляет изменение в систему контроля версий, он запускает рабочий процесс конвейера для проверки изменений в среде тестирования. После прохождения интеграционных и сквозных тестов конвейер может остановиться в ожидании ручного одобрения, прежде чем развертывать изменения в промышленной среде.

Почему нужно использовать CD вместо CI? Процесс непрерывной поставки включает в себя все автоматические тесты, над написанием которых вы усердно работали в главе 6. Кроме того, конвейер поставки включает в себя входной контроль качества. У коллеги по команде, проверяющего ваше изменение, будет больше уверенности в том, что оно работает правильно.

ПРИМЕЧАНИЕ Подробное описание технологии непрерывной поставки займет целую книгу! Я применила технологию непосредственно к инфраструктуре и попыталась раскрыть эту тему в одном разделе. Для любителей более практичных примеров я создала пример конвейера, доступный по адресу <http://mng.bz/mOy8>. Конвейер использует GitHub Actions для развертывания службы hello-world в Google Cloud Run. Его этапы содержат модульные тесты, развертывание среды тестирования и интеграционные тесты.

Применение CI/CD подразумевает небольшие и частые изменения кода. Вы автоматически отправляете эти изменения в тестовую среду и ждете ручного одобрения, прежде чем отправлять их в производство. Однако изменения, ожидающие одобрения вручную, накапливаются, как пробка на дороге в час пик. Несколько машин, вставших в узком месте, замедляют движение транспорта, идущего следом, иногда на несколько часов!

Обычно при ручном утверждении формируется пакет изменений, что создает некоторые проблемы. Если вы отправляете большую партию изменений (рис. 7.5), то приходится ждать, пока система работает и развернет изменения. Это может привести к проблемам, поскольку некоторые изменения конфликтуют. В итоге ваша команда рискует потратить несколько дней на то, чтобы выяснить, какая комбинация изменений привела к сбою.

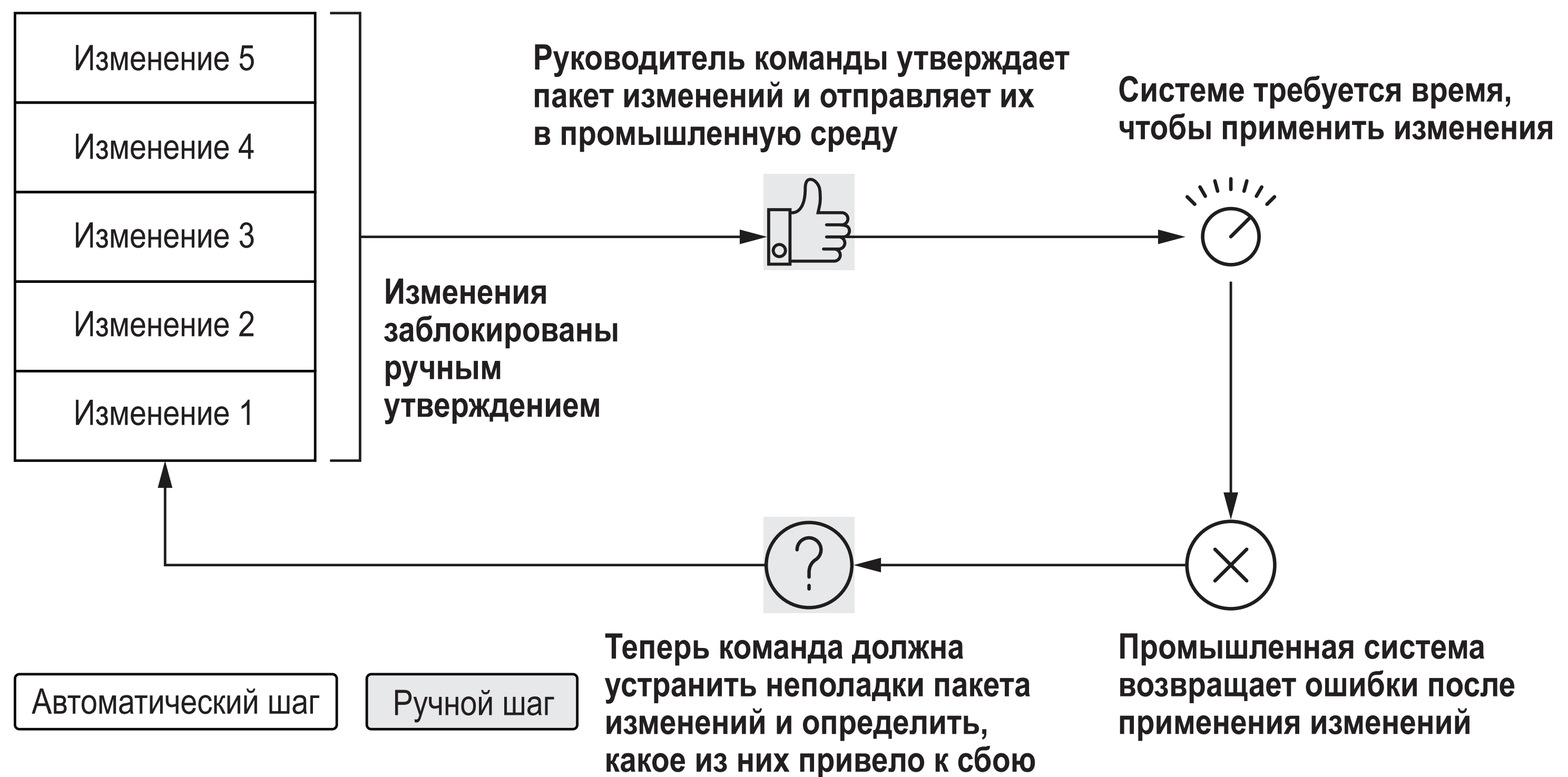


Рис. 7.5 При ручном утверждении избегайте отправки больших пакетов изменений в промышленную систему, чтобы предотвратить случаи, когда приходится вручную устранять проблемы

При использовании технологии CI/CD одобряйте изменения как можно быстрее. Внедрите более короткий цикл обратной связи для ручной проверки. Можно также ограничить количество изменений, одобряемых одновременно. Оба решения снижают часть риска, связанного с ручными проверками. В следующем разделе я расскажу о другом решении, которое полностью исключает этап ручного одобрения.

7.1.3 Непрерывное развертывание

Можно ли избавиться от больших пакетов изменений, исключив этап ручной проверки в конвейере поставки? Да, можно! Но сначала вам нужно лучше разобраться в технологиях CI/CD.

Удаление этапа ручной проверки из вашего конвейера означает, что мы должны быть уверены в надежности тестирования. В конвейер на рис. 7.6 добавлены дополнительные интеграционные и сквозные тесты для проверки системы и предусмотрена автоматическая отправка изменений в промышленную среду. Теперь мы ожидаем, что тесты достаточно хорошо проверяют функциональность системы, а внесенные изменения можно легко отменить. Мы убираем этап ручного одобрения и сразу отправляем обновления в производство.

Непрерывное развертывание исключает необходимость ручной проверки и переносит изменения непосредственно из тестовой среды в рабочую.

ОПРЕДЕЛЕНИЕ Механизм *непрерывного развертывания* (continuous deployment) IaC развертывает и тестирует изменения вашей инфраструктуры в тестовой среде и автоматически выпускает изменения в рабочую среду после прохождения тестов.



Рис. 7.6 Механизм непрерывного развертывания полностью автоматизирует тестирование и отправку изменений в производство

Автоматическое развертывание предотвращает задержки на пути обновлений. Внесение изменений в инфраструктуру часто занимает часы и влияет на многочисленные зависимости. Применяйте технологию непрерывного развертывания к своей инфраструктуре, если у вас есть проработанная стратегия тестирования и навыки устранения сбоев.

Использование методов устранения сбоев, описанных в главе 11, поможет вам применять непрерывное развертывание на практике. Однако большинство организаций не полностью поддерживают непрерывное развертывание своей инфраструктуры, поскольку не уверены в результатах тестирования или возможности отката изменений. Чтобы приблизиться к модели непрерывного развертывания, необходимо иметь хороший опыт тестирования и отката обновлений.

7.1.4 Выбор способа поставки

Непрерывная поставка и развертывание вместе образуют рабочий процесс тестирования и поставки IaC в промышленную среду. Но не следует ожидать, что вы будете чувствовать себя в безопасности, автоматизируя все изменения! Я рекомендую применять для управления изменениями инфраструктуры сочетание непрерывной поставки и развертывания. Но прежде чем выбирать подход к реализации, необходимо классифицировать типы изменений, с которыми вам предстоит работать.

Типы изменений инфраструктуры

От типа изменения зависит способ его поставки в производство. Необходимо сначала провести подробный анализ, чтобы составить или получить внутреннюю классификацию типов изменений и автома-

тизировать тестирование и проверку каждого из них. В противном случае вы рискуете нарушить требования проверки изменений.

Представьте, что вы каждую неделю вносите стандартные изменения на сервер. Вы обновляете IaC сервера новым тегом. Автоматизация никогда не меняется и не часто дает сбои. Когда что-то выходит из строя, вы точно знаете, как это исправить. Регулярное обновление сервера становится хорошим кандидатом на применение подхода CD.

На рис. 7.7 показано, что вы постоянно развертываете обновление сервера непосредственно в промышленной среде (без ручной проверки). Конвейер заменяет этап ручной проверки после тестовой среды на тест для проверки префикса в сообщении коммита. Обновление вашего сервера содержит сообщение коммита со словом `standard`, поэтому конвейер не нуждается в ручном одобрении.

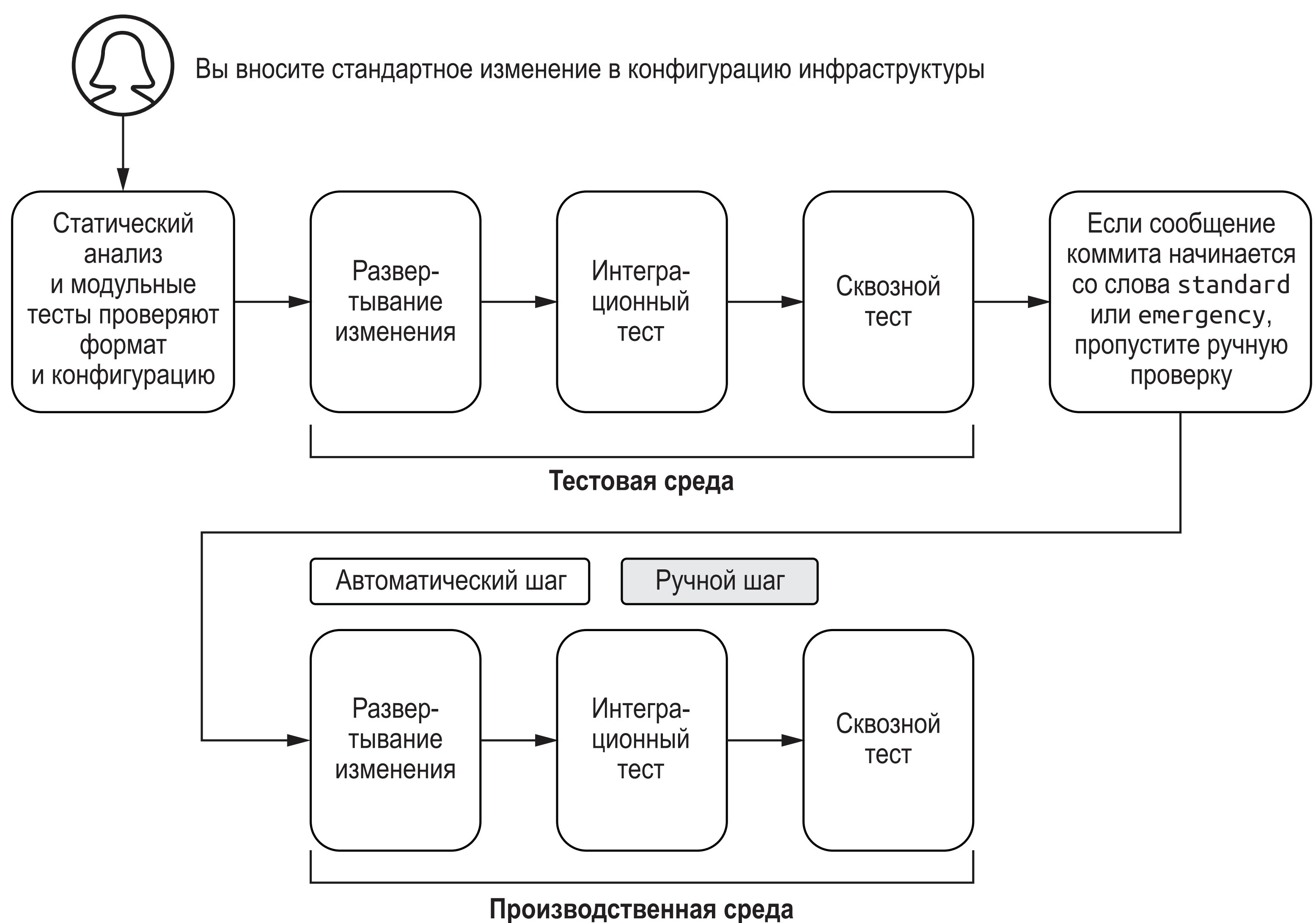


Рис. 7.7 Стандартное или срочное изменение может пройти первичную проверку, прежде чем ваш конвейер поставки автоматически отправит его в промышленную среду

Вы регулярно вносите *стандартные изменения* в свою инфраструктуру. К таким изменениям, например, относятся обновление образов контейнеров в Kubernetes, развертывание новой очереди или добавление нового оповещения в систему мониторинга. Если изменение не удалось, вы можете отменить его, не влияя на остальную инфраструктуру.

ОПРЕДЕЛЕНИЕ *Стандартное изменение* инфраструктуры – это часто реализуемое изменение с четко определенным поведением и планом отмены.

Почему я рекомендую начинать с непрерывного развертывания стандартных изменений? Стандартное изменение часто представляет собой автоматизированное, четко определенное исправление. Членам вашей команды нет необходимости вручную одобрять повторяющиеся изменения. Стандартное изменение отвлекает их от более важных дел.

Другие типы изменений тоже выигрывают от непрерывного развертывания. Допустим, вы обнаружили, что приложение на сервере перестало работать. Вам нужно, чтобы приложение снова заработало как можно скорее. Вместо того чтобы вносить стандартные изменения, вы реализуете исправление в IaC и обновляете сообщение коммита, добавляя в него слово *emergency* (неотложный). После отправки изменения ваша система сборки обходит стадию ручного одобрения, поскольку сообщение коммита определяет изменение как неотложное.

Помимо непрерывной поставки стандартных изменений, вы можете выбрать непрерывную поставку фиксов для экстренных ситуаций, когда нужно восстановить работоспособность системы. Если возможно, публикуйте исправления, используя IaC и сообщение коммита, чтобы выделить срочное изменение.

ОПРЕДЕЛЕНИЕ *Срочное изменение (фикс)* инфраструктуры – это обновление, которое необходимо как можно скорее выпустить в производство, чтобы восстановить работоспособность системы.

Экстренные изменения применяются в промышленной среде без ручной проверки, поскольку часто требуется быстро исправить систему. Ручная проверка может задержать решение проблемы. Отказ от одобрения срочных изменений поможет вам быстро решить проблему и сохранить историю ее решения.

Чтобы иметь возможность постоянно применять стандартные и срочные изменения в обход ручной проверки, в конвейере должно быть предусмотрено автоматизированное тестирование. Кроме того, вам необходимо стандартизировать структуру сообщения коммита, позволяющего обходиться без ручной проверки. Наличие обходного пути позволяет инженерам развертывать срочные исправления без очереди, а команды, контролирующие соответствие и безопасность, сохраняют возможность видеть последовательность изменений.

Можно ли просто внести срочное изменение вручную?

Я настоятельно рекомендую вам использовать IaC и конвейер поставки для внесения срочных изменений. В коммитах сохраняется история ваших действий по устранению проблем, и конвейер проверяет ваши изменения, прежде чем запускать автоматизацию, которая потенциально может ухудшить работу системы.

Однако может возникнуть ситуация, когда работа конвейера развертывания занимает недопустимо много времени. В таком случае можно прибегнуть к внесению изменений вручную, четко осознавая, что автоматические тесты и проверки, выполняемые конвейером, скорее навредят, чем принесут пользу.

После внесения изменений вручную согласуйте фактическое состояние инфраструктуры с описанием в репозитории. Практика согласования предполагает ручное обновление конфигурации в соответствии с ресурсом инфраструктуры (об этом методе рассказывается в главе 2).

Некоторые изменения принципиально *не* должны использовать непрерывное развертывание. Представьте, что вам поручили новый проект. Вам необходимо включить IPv6 во всех сетях. Обновление сети может негативно повлиять на каждое приложение и систему в целом!

Для столь значимого обновления не стоит пропускать этап ручной проверки. Вам нужна помощь квалифицированных сетевых инженеров, которые проверят ваш IaC. В соответствии с процессом на рис. 7.8 вы обновляете IaC своих сетей до IPv6 и ожидаете одобрения перед отправкой в промышленную среду. Сам факт запроса одобрения сообщает другим смежным командам и инженерам, что ваши изменения могут причинить существенный ущерб системе в случае неудачи.

Нестандартные или существенные обновления могут повлиять на архитектуру, безопасность или доступность системы. Эти изменения нуждаются в заявке с обоснованием и обсуждением. Они также подразумевают ручную проверку изменений вашими коллегами по команде или компании.

ОПРЕДЕЛЕНИЕ *Нестандартное или существенное обновление* инфраструктуры потенциально влияет на архитектуру, безопасность или доступность системы. Такие обновления не имеют четко определенных планов реализации или отката.

Существенные обновления могут иметь значительное влияние на систему или высокий риск неудачи. Аналогично новые или неиз-

вестные изменения могут привести к непредсказуемым результатам и сложным шагам отката. Запрос на одобрение сигнализирует о том, что вам может понадобиться помощь, если изменения не удастся внести. Примерами новых или серьезных изменений могут служить обновления сетевых блоков CIDR (если они влияют на другие распределения), DNS, изменения сертификатов, обновления кластеров Kubernetes или рефакторинг платформы (например, добавление менеджера секретов в приложения).

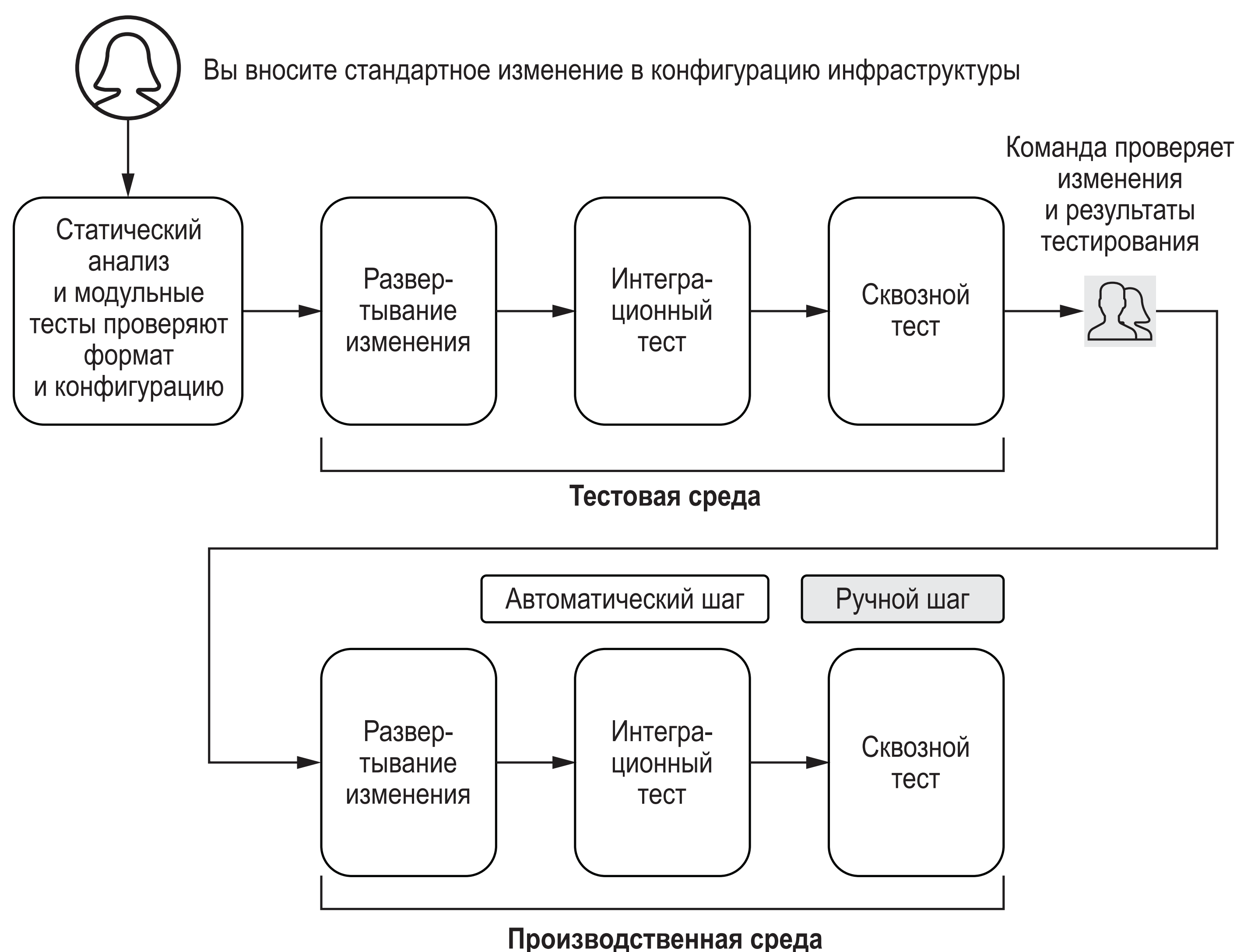


Рис. 7.8 Нестандартные или существенные обновления должны пройти ручное одобрение коллег перед применением в производстве

Подход, основанный на типе изменения

После классификации вносимых вами изменений по типам вы можете принять решение о выборе способа поставки. По большей части для стандартных и срочных изменений выбирают непрерывное развертывание, тогда как для нестандартных и существенных изменений чаще используют непрерывную поставку.

В табл. 7.1 представлены некоторые типы изменений, подход к их реализации и примеры. Однако из этих общих практик есть исключения. Некоторые стандартные изменения могут потребовать непрерывной поставки, поскольку они влияют на другие ресурсы. Напротив, изменения, связанные с только что созданными средами, могут реализовать подход непрерывного развертывания, поскольку он не влияет на другие системы.

Таблица 7.1. Типы обновлений инфраструктуры и подход к реализации

Тип обновления	Способ поставки	Одобрение перед производством	Пример
Стандартное	Непрерывное развертывание	Нет	Добавление сервера при масштабировании системы
Срочное	Непрерывное развертывание или ручное обновление	Нет	Откат образа операционной системы к предыдущей версии
Существенное	Непрерывная поставка	Да	Включение SSL для всех служб и инфраструктуры
Нестандартное	Непрерывная поставка	Да	Развертывание нового типа компонента инфраструктуры

Непрерывная интеграция, поставка и развертывание также применимы к жизненному циклу разработки программного обеспечения. Однако применение этих концепций к жизненному циклу инфраструктуры расширяет границы процессов изменений и анализа в вашей организации. Регулярная категоризация ваших изменений и оценка процесса изменений и тестирования способствует достижению баланса между производительностью и управляемостью, о чем я упоминала в рамках практики совместного использования модулей.

Управление конфигурацией

Инструменты управления конфигурацией должны использовать аналогичный подход для оценки типов изменений и применения непрерывной поставки или развертывания.

Возьмите за правило быстро просматривать и одобрять изменения и как можно скорее внедряйте их в производство. Чем больше пакет изменений, тем больше потенциальный ущерб, если что-то вдруг пойдет не так. Если вы выпускаете все обновления одним пакетом и нарушаете работу критически важного для бизнеса приложения, вам придется в аварийной ситуации проверять все обновления в пакете. Сложность устранения неполадок возрастает, когда необходимо определить, какое изменение повлияло на систему.

Упражнение 7.1

Выберите стандартное изменение инфраструктуры в вашей организации. Что вам нужно для того, чтобы уверенно и непрерывно внедрять изменения в промышленную среду? Как насчет непрерывного развертывания? Схематически изобразите этапы ваших конвейеров поставки в обоих случаях.

Ответы на упражнения даны в приложении В.

7.1.5 Модули

Как устроен конвейер поставки модулей? О совместном использовании, выпуске и управлении версиями модулей инфраструктуры вы узнали в главе 5, а о тестировании их функциональности – в главе 6. Я упомянула идею автоматизации процесса тестирования и выпуска модулей, но не объяснила полностью процесс их поставки.

Процессы поставки инфраструктурных модулей немного отличаются от примеров, которые я описала для производственной конфигурации. Конвейер поставки должен выглядеть, как показано на рис. 7.9, чтобы выпускать модуль после тестирования, а не сразу поставлять его в промышленную среду. В этом конвейере сохранен этап ручной проверки, чтобы ваша команда была в курсе изменений.

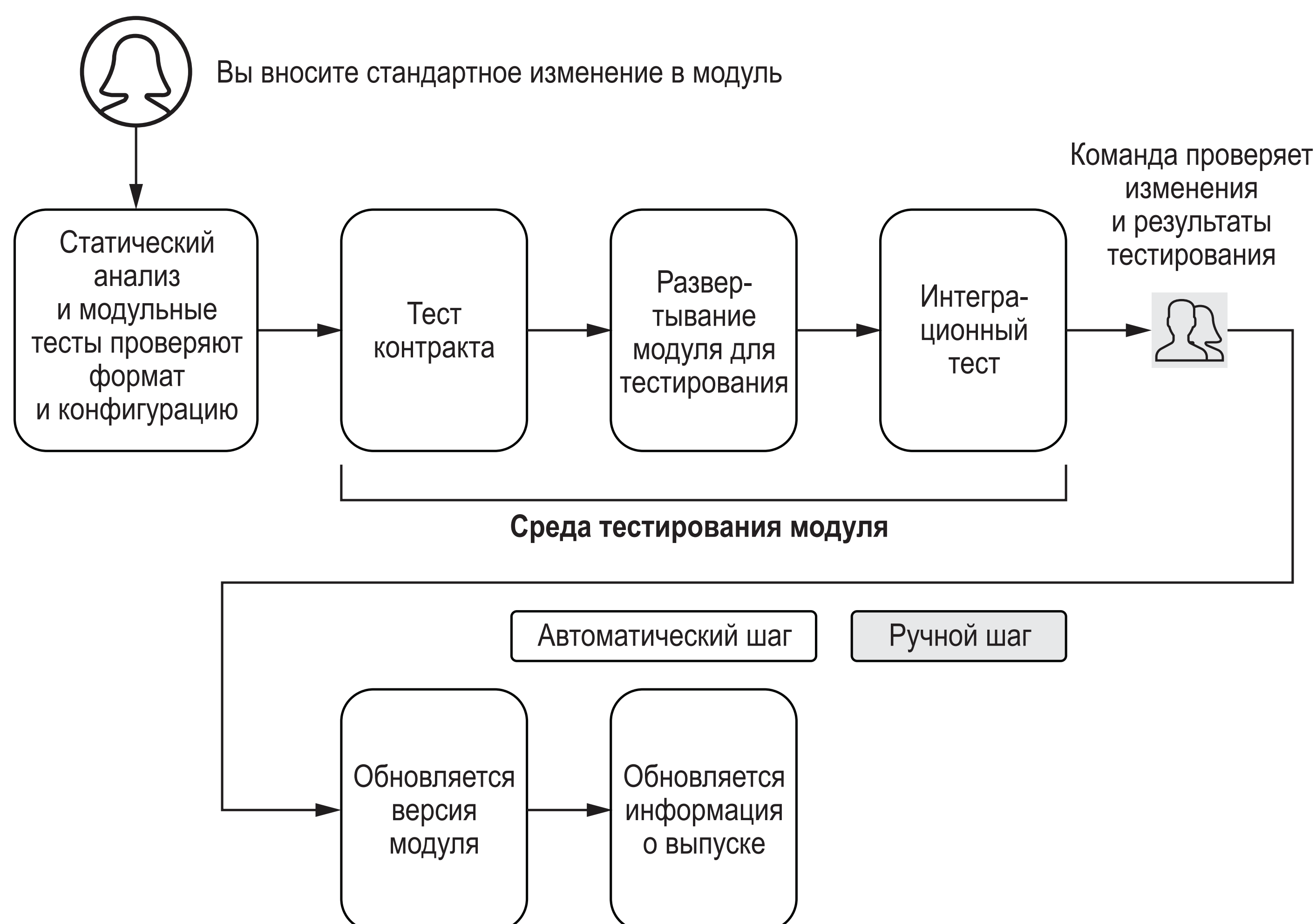


Рис. 7.9 После тестирования модуля подождите, пока команда рассмотрит изменения и результаты тестирования, прежде чем обновлять и выпускать новую версию модуля

Для классификации изменений модулей подходят те же типы, что и для изменений конфигурации, включая стандартные, экстренные, существенные и нестандартные обновления. В табл. 7.2 описаны типы изменений, подходы к их реализации и примеры. По большей части они соответствуют подходу, рекомендованному для конфигураций.

Некоторые изменения в модулях, например обновления конфигурации базы данных или другой специализированной инфраструктуры, включающей данные, нуждаются в консультациях или участии

в разработке кода со стороны эксперта в предметной области. Однако к срочным изменениям в модулях можно применить другой способ поставки.

Таблица 7.2. Типы обновлений модуля и подход к поставке

Тип обновления	Способ поставки	Одобрение перед производством	Пример
Стандартное	Непрерывное развертывание	Нет	Добавление возможности менять параметры по умолчанию
Срочное	Непрерывное развертывание или ветвление	Нет	Откат образа операционной системы к предыдущей версии
Существенное	Непрерывная поставка	Да	Обновление базы данных или инфраструктуры под новый тип данных
Нестандартное	Непрерывная поставка	Да	Развертывание нового модуля сервера

Срочные изменения модулей означают создание новой версии модуля, куда войдут исправления. Добиться выделения версии можно одним из двух способов. Вы можете внести исправление и продолжать развертывать и выпускать новую версию модуля с изменениями. Альтернативно вы можете создать ветку репозитория модулей и обновить конфигурацию инфраструктуры, чтобы она ссылалась на эту ветку.

ОПРЕДЕЛЕНИЕ Ветка (branch) в системе контроля версий – это указатель на снимок кода. Она позволяет вам отдельно вносить изменения на основе этого снимка.

После тестирования ветки вы можете обновить основную ветку модуля стандартным изменением. Ветвление модуля помогает быстро реализовать срочное изменение и позднее согласовать изменения модуля в обычном порядке.

Если я знаю, что другие команды фиксируют свои версии, я предпочитаю непрерывное развертывание новой версии модуля с обновлением. Хотя ветка может отделить срочное исправление, мне нужно не забыть впоследствии объединить ее обратно с основной версией модуля. В следующем разделе вы узнаете о моделях ветвления и о том, как их применять к изменениям IaC.

Построение образа и управление конфигурацией

Конвейеры сборки Docker образов и модули используют один и тот же подход. Мы всегда версионируем и тестируем изменения в образах перед их развертыванием в промышленной среде.

7.2 Модели ветвления

Помимо реализации непрерывной поставки или развертывания, вам необходимо стандартизировать способ слияния изменений с вашей основной конфигурацией. Главная ветка системы контроля версий служит источником достоверной информации о конфигурации. Обновление конфигурации требует дополнительной координации и сотрудничества внутри вашей команды.

Представьте, что вы хотите ограничить доступ к брандмауэру, пока ваш товарищ по команде обновляет его лицензию. У вашей команды есть конвейер непрерывной поставки для тестирования и ручного утверждения изменений в промышленном межсетевом экране. Однако изменения, вносимые вами и вашим товарищем по команде, создают две проблемы.

Во-первых, как вам организовать совместную работу и тестирование изолированных изменений? Во-вторых, как понять, какое изменение должно произойти первым? На рис. 7.10 показана дилемма, возникшая перед вами и вашим товарищем по команде: кто должен развернуть изменения первым. Вам нужно избежать одновременно-го внесения обоих изменений. Если брандмауэр вызовет сбой доступа к сети, вы не будете знать, какое изменение вызвало проблему.

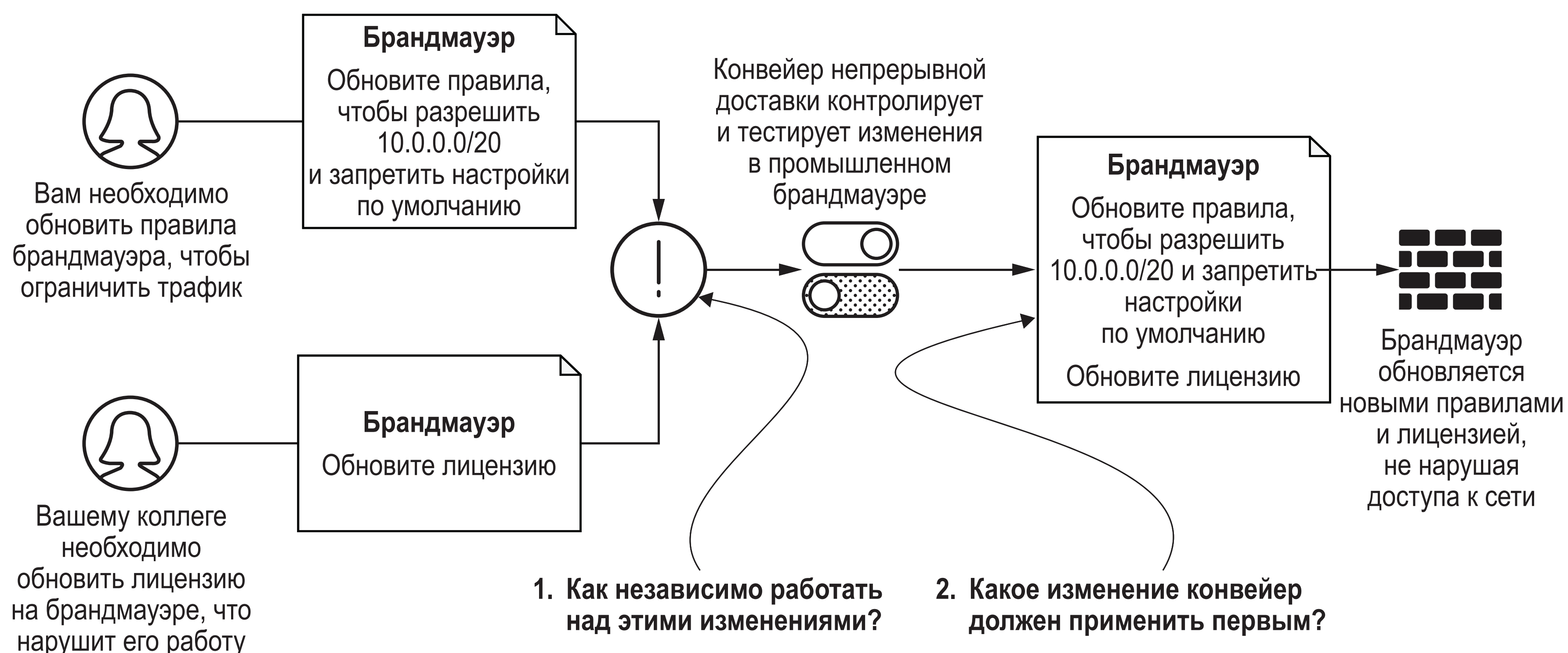


Рис. 7.10 Даже если у вас есть конвейер CD для поставки изменений в брандмауэр, вам потребуется дополнительная координация действий в команде, чтобы определить, какое изменение должно применяться в первую очередь

От модели ветвления зависит то, как ваша команда использует контроль версий, чтобы обеспечить параллельную работу, сводя к минимуму сбои и сложность устранения неполадок. Вы можете выбрать одну из двух моделей ветвления: *функционально ориентированную разработку* (feature-based development) или *магистрально ориентированную разработку* (trunk-based development).

ОПРЕДЕЛЕНИЕ Модель ветвления (branching model) определяет, как ваша команда использует систему контроля версий для обеспечения параллельной работы и разрешения конфликтов в своей деятельности.

Каждая модель ветвления имеет свои сложности в реализации, особенно в IaC. Я опишу применение двух вариантов модели на примере координации разработки правил брандмауэра и изменения лицензии между вами и вашим товарищем по команде. Затем я расскажу о некоторых ограничениях каждого подхода и о том, как выбрать наиболее подходящий вариант для вашей команды.

7.2.1 Функционально ориентированная разработка

Что, если вы и ваш товарищ по команде сможете работать над изменениями изолированно, прежде чем объединять их? Если ваш коллега создает ветку с изменением лицензии, а вы создаете ветку с изменениями брандмауэра, вы изолируете свои обновления друг от друга. Когда вы оба закончите, вы объедините свои изменения с основной веткой main и разрешите конфликты обновлений.

На рис. 7.11 показана координация изменений в разных ветках между вами и вашим коллегой. Вы называете свою ветку для правила брандмауэра TICKET-002, а ваш товарищ по команде дает своей ветке для обновления лицензии имя TICKET-005. Изменения правил брандмауэра сначала утверждаются экспертами, поэтому вы помещаете их в ветку main и развертываете в рабочей среде. Ваш коллега продолжает работать над обновлением лицензии. Он подтягивает ваше обновление правил брандмауэра в свою ветку для дальнейшего тестирования, прежде чем объединить свои изменения обратно с главной веткой.

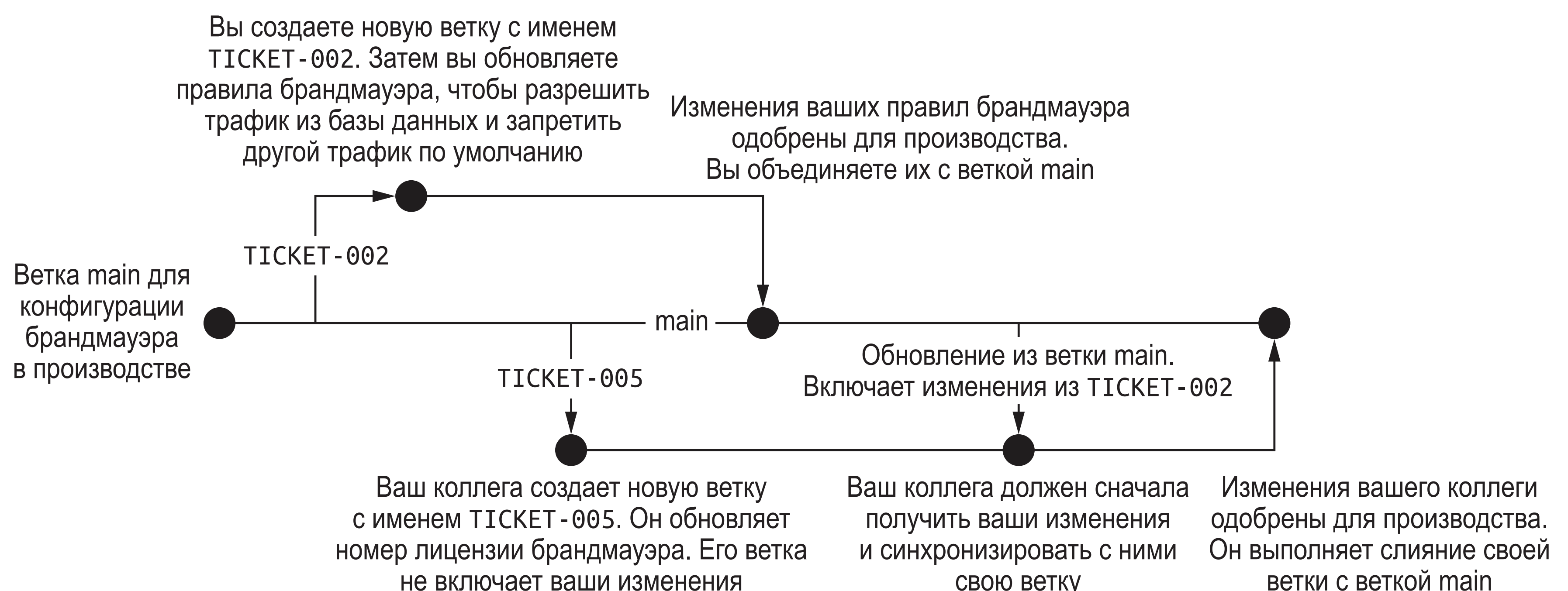


Рис. 7.11 При функционально ориентированной разработке вы изолируете изменения в своей ветке и разрешаете конфликты с главной конфигурацией

Функционально ориентированная разработка позволяет вам развивать изменения независимо от ваших товарищей по команде, изолируя эти изменения в отдельной ветке.

ОПРЕДЕЛЕНИЕ *Функционально ориентированная разработка* (также известная как *функциональное ветвление*, или *Git Flow*) – это модель ветвления, которая разделяет различные изменения на отдельные ветви. После тестирования вы объединяете изменения в конкретной ветке с основной конфигурацией.

Функциональное ветвление помогает сосредоточиться на своих изменениях независимо от других разработчиков. Однако вам необходимо постоянно получать изменения из главной ветки и синхронизировать с ними свою ветку. Функционально ориентированная разработка работает лучше всего, когда каждый член команды старательно обновляет и тестирует свои ветки.

Давайте рассмотрим функциональное ветвление в действии. Представьте, что вы начинаете рабочий процесс разработки правил брандмауэра с клонирования локальной копии конфигурации из системы контроля версий:

```
$ git clone git@github.com:myorganization/firewall.git
```

Вы создаете ветку, которая содержит указатель на ваши обновления. Я рекомендую называть ветку по номеру заявки (например, TICKET-002), связанной с изменением, хотя вы также можете использовать описательное имя, разделенное дефисом:

```
$ git checkout -b TICKET-002
```

Вы вносите изменения в правила брандмауэра в ветке. Затем вы используете командную строку, чтобы сделать коммит изменений в локальной ветке:

```
$ git commit -m "TICKET-002 Only allow traffic from database"
[TICKET-002 cdc9056] TICKET-002 Only allow traffic from database
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 firewall.py
```

Теперь у вас есть локальные изменения, но вы хотите, чтобы их проверили коллеги. Вы отправляете изменения в удаленную ветку:

```
$ git push --set-upstream origin TICKET-002
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 1.06 KiB | 1.06 MiB/s, done.
```



```
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:myorganization/firewall.git
 * [new branch]      TICKET-002 -> TICKET-002
Branch 'TICKET-002' set up to track remote branch
➡ 'TICKET-002' from 'origin'.
```

Тем временем ваш товарищ по команде работает над изменением TICKET-005, который обновляет лицензию. Он создает новую ветку под названием TICKET-005 с изменениями, которые не включают обновления правил брандмауэра. Обратите внимание, что ваша ветка не включает обновленную лицензию, а его ветка не включает ваши обновленные правила брандмауэра. Вы можете просмотреть различия между двумя ветками:

```
$ git diff TICKET-002..TICKET-005
diff --git a/firewall.py b/firewall.py
index 74daecd..aaf6cf4 100644
--- firewall.py
+++ firewall.py
@@ -1,3 +1,3 @@
-print("License number is 1234")
+print("License number is 5678")

-print("Firewall rules should allow from database and deny by default.")
\ No newline at end of file
+print("Firewall rules allow all.")
\ No newline at end of file
```

Вы открываете *запрос на изменение*, уведомляя свою команду о том, что вы завершили внесение изменений.

ОПРЕДЕЛЕНИЕ *Запрос на изменение* (pull request) уведомляет ответственных за сопровождение репозитория о наличии внешних изменений, которые вы хотели бы объединить с основной конфигурацией.

Вы добавляете ответственных за согласование изменений для рассмотрения вашего запроса на включение. Они одобряют изменения, и вы объединяете свою ветку с основной.

Ваш коллега еще не получил одобрения на обновление лицензии. Чтобы гарантировать, что обновление не повлияет на производственную конфигурацию, ему необходимо получить все изменения из ветки main, включая ваши изменения в TICKET-002:

```
$ git checkout main
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
```



```
$ git pull --rebase
Updating 22280e7..084855a
Fast-forward
 firewall.py | 2 +-
 1 file changed, 1 insertion(+)
```

Затем он возвращается в свою ветку с именем TICKET-005 и включает изменения из ветки main в ветку TICKET-005:

```
$ git checkout TICKET-005
Switched to branch 'TICKET-00B'
Your branch is up to date with 'origin/TICKET-005'.
```

```
$ git merge main
Auto-merging firewa11.py
Merge made by the 'recursive' strategy.
 firewall.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Когда ваш коллега рассмотрит конфигурацию брандмауэра, он найдет изменения, которые вы внесли в TICKET-002. Он может обновить свою ветку изменениями из ветки main:

```
$ git push --set-upstream origin TICKET-005
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 1.06 KiB | 1.06 MiB/s, done.
5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:myorganization/firewall.git
 * [new branch] TICKET-005 -> TICKET-005
Branch 'TICKET-005' set up to track remote branch 'TICKET-005' from 'origin'.
```

Как только изменения вашего коллеги получают одобрение, он сможет объединить новую лицензию брандмауэра с веткой main.

Функционально ориентированная разработка требует от каждого члена команды выполнения множества шагов. Для упрощения рабочего процесса автоматизируйте процессы тестирования и слияния. Используйте конвейер поставки для организации изменений по веткам.

На рис. 7.12 показан рабочий процесс конвейера поставки при функционально ориентированной разработке для двух участников. Вы и ваш коллега получаете каждый по ветке конвейера со своей собственной средой тестирования. Например, у вас есть ветка с именем TICKET-002 и новая среда брандмауэра, которая изолирует ваши изменения. Вы запускаете модульные тесты, развертываете изменения, а также запускаете интеграционные и сквозные тесты в среде брандмауэра TICKET-002.

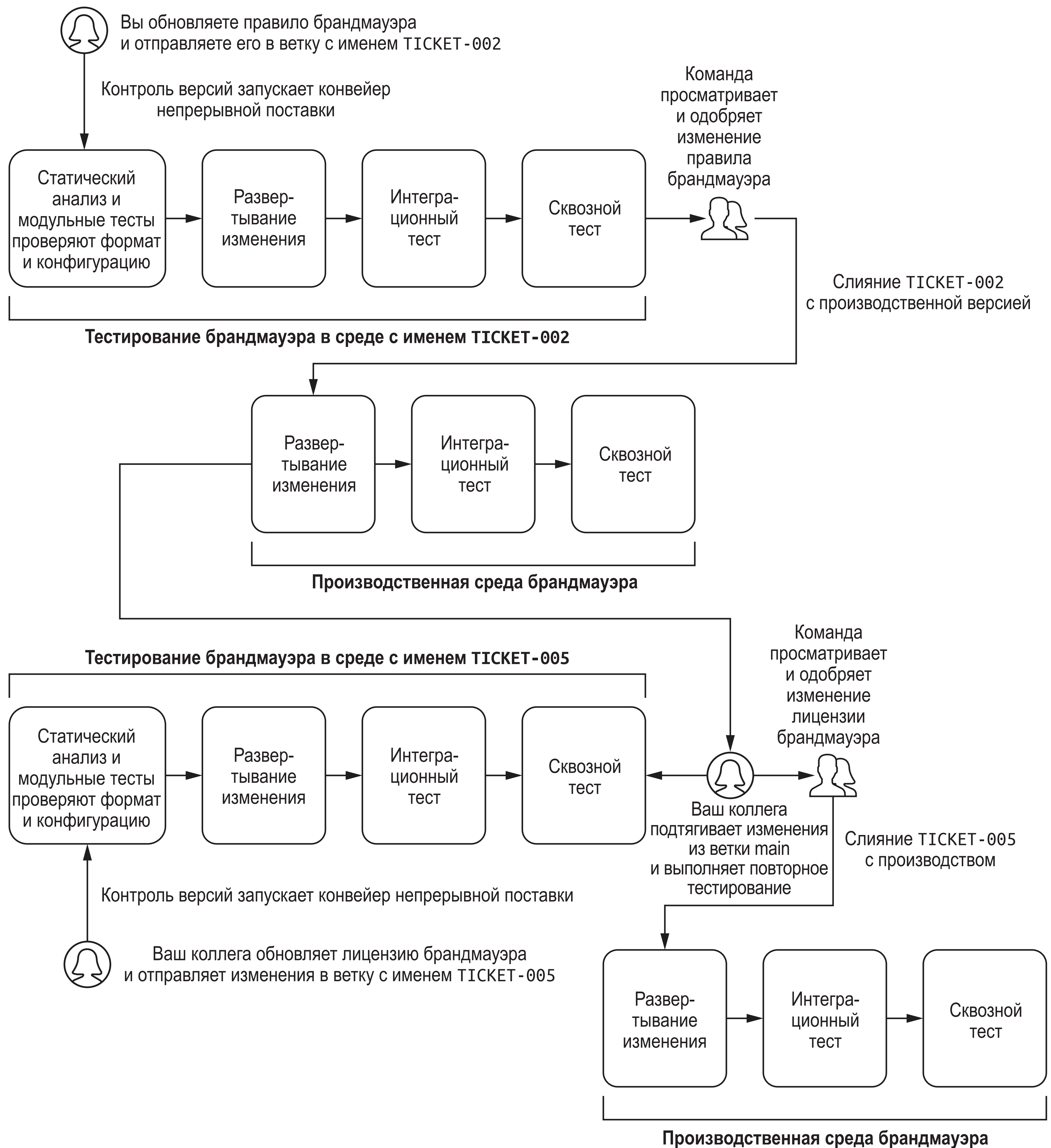


Рис. 7.12 Применение функционально ориентированной разработки для изолированного тестирования ваших изменений в ветке

Если тестирование вашей ветки прошло успешно, вы объединяете изменения с веткой main. Пока вы работали над изменением, ваш коллега отдельно создал собственную ветку и среду брандмауэра под названием TICKET-005. Ваш товарищ по команде понимает, что вы недавно обновили конфигурацию брандмауэра.

На рис. 7.12 показано, что ваш коллега получает изменения из ветки main, и проверяет, что эти изменения по-прежнему работают с его веткой и средой. После того как ваш коллега выполнит те же модульные, интеграционные и сквозные тесты в своей ветке, он

объединит изменения TICKET-005 с веткой `main` для развертывания в производстве.

Зачем создавать тестовую среду для каждой ветки? Такой подход позволяет изолировать ваши изменения и протестировать их относительно основной ветки. Тестовая среда ветки создается на время и под конкретную задачу, что помогает свести к минимуму потребность в постоянной тестовой среде и снизить общую стоимость инфраструктуры. Однако создание тестовой среды может занять некоторое время.

Функционально ориентированная разработка имеет свои преимущества, в том числе:

- возможность изолировать изменения в ветке;
- возможность тестировать изменения внутри ветки;
- неявный этап экспертной оценки. Вы можете объединить свои изменения с рабочей версией только в том случае, если кто-то их одобрит;
- разделение срочных изменений по ветке. После проверки срочного изменения вы можете объединить его с веткой `main`.

К счастью, популярные службы хостинга репозиторий (например, GitHub или GitLab) помогают нам автоматизировать модель функционально ориентированной разработки. В механизм автоматизации входят метки для отслеживания определенных функций, проверка статуса с помощью интеграционных тестов перед объединением вашей ветки и автоматическое удаление старых веток. Вы также можете определить список рецензентов и автоматически добавлять их в запросы на включение.

7.2.2 Магистрально ориентированная разработка

Представьте, что ваша организация не хочет создавать среду тестирования для каждой ветки, а многие сотрудники считают некомфортным для себя рабочий процесс функционально ориентированной разработки. Вместо создания боковой ветки вы и ваш коллега работаете вместе над основной веткой `main` (магистраль процесса).

На рис. 7.13 показан механизм совместной работы в ветке `main`. Вы обновляете правило брандмауэра и отправляете изменения в удаленный репозиторий. Затем ваш коллега обновляет свой локальный репозиторий, включая ваши изменения, и отправляет свои изменения в ветку `main`.

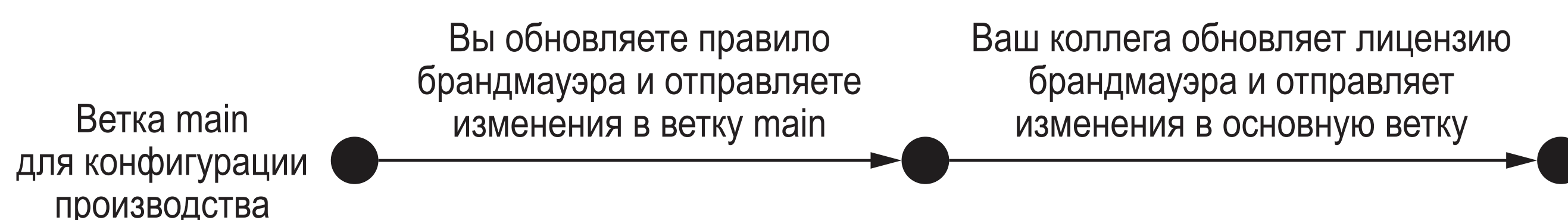


Рис. 7.13 При использовании магистрально ориентированной разработки вы поддерживаете только ветку `main` и обновляете производственную конфигурацию напрямую

Рабочий процесс выглядит проще, поскольку вы оба переходите в одну ветку. Магистрально ориентированная разработка означает, что вы отправляете изменения непосредственно в ветку `main`, не создавая ветку на каждое изменение версий.

ОПРЕДЕЛЕНИЕ *Магистрально ориентированная разработка* – это модель ветвления, согласно которой все изменения передаются непосредственно в ветку `main`. В этой модели принято вносить небольшие обновления и использовать тестовую среду для проверки успеха изменений.

Магистрально ориентированная разработка не позволяет вам вносить изменения независимо от ваших товарищей по команде. Однако это ограничение становится преимуществом. Такая модель заставляет вас вносить изменения *в определенном порядке*. Вы можете быстро определить, какой коммит вызывает неудачное изменение, и устранить его. Таким образом, модель предлагает продуманный способ организации и применения изменений IaC.

Давайте применим магистрально ориентированную разработку к вашему правилу брандмауэра и обновлениям лицензии брандмауэра вашего товарища по команде. Вы начинаете с клонирования локальной копии конфигурации брандмауэра из системы контроля версий. При клонировании конфигурации вы можете проверить текущую ветку:

```
$ git clone git@github.com:myorganization/firewall.git
$ git branch --show-current
main
```

Вы вносите изменения правил брандмауэра в ветке `main` и делаете коммит изменений:

```
$ git commit -m "TICKET-002 Only allow traffic from database"
[TICKET-002 cdc9056] TICKET-002 Only allow traffic from database
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 firewall.py
```

Обновите локальную копию, чтобы убедиться, что вы получаете новые изменения из ветки `main`. Используйте команду `git pull --rebase`, чтобы получить изменения из удаленного репозитория, объединить их с локальной копией и перебазировать локальную историю:

```
$ git pull --rebase
Already up to date.
```

Теперь вы можете отправить изменения в основную ветку `main`. Ваша команда `push` должна запустить конвейер поставки, показанный на рис. 7.14. Этот конвейер выполняет модульные и интеграционные тесты в тестовой среде. После прохождения всех этапов тест-

тирования конвейер ожидает ручного одобрения вашей команды. Ваш коллега может просмотреть ваши изменения и одобрить их. Как только он одобрит ваше изменение, конвейер обновит правила брандмауэра в промышленной среде.

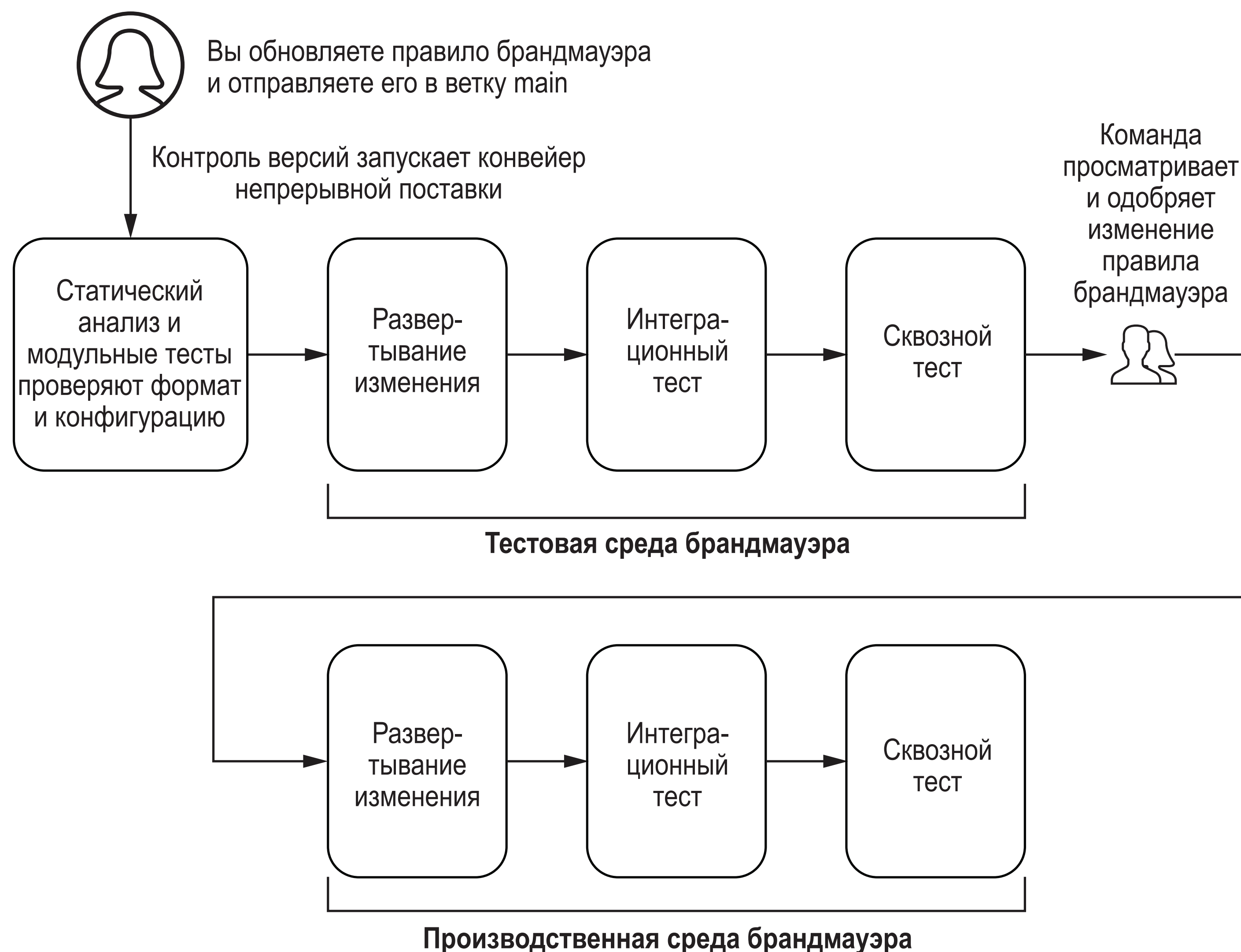


Рис. 7.14 Магистрально ориентированная разработка нуждается в конвейере непрерывной поставки обновлений в промышленную среду

Если изменение правила брандмауэра не удалось, конвейер поставки останавливается до развертывания в производстве. Вы заметите сбой на этапе тестовой среды и отмените изменения. Все остальные разработчики могут продолжить внесение изменений в рабочую версию, пока вы устраняете свою ошибку.

Магистрально ориентированная разработка во многом зависит от конвейеров поставки для тестирования и развертывания изменений. В конвейер поставки должна входить постоянная среда для тестирования конфликтов между изменениями. Хотя постоянная тестовая среда требует затрат, она более точно отражает поведение изменений в промышленной среде.

Рабочий процесс магистрально ориентированной разработки состоит из очень небольшого количества шагов. Большинству работающих с инфраструктурой команд нравится этот процесс внесения изменений, поскольку он строго упорядочивает обновления. Магистрально ориентированная разработка создает непрерывный цикл обратной связи о том, как различные изменения влияют друг на друга. Она также поддерживает практику внесения незначительных изменений, разрешения конфликтов обновлений в основной

ветке и непрерывной поставки обновлений в рабочую среду. Если изменения вашей команды конфликтуют, вы можете быстро определить, какие зависимости повлияли на тестовую среду.

Однако эта модель требует навыков разрешения конфликтов обновлений и дисциплины для согласования изменений. Вы не изолируете свои изменения, и наличие только одной ветки может затруднить совместную работу. Но если вы сможете избежать проблем совместной работы, то быстро обнаружите, что магистрально ориентированная разработка обеспечивает лучшую видимость изменений IaC в вашей команде.

7.2.3 *Выбор модели ветвления*

Я провела много часов с командами разработчиков программного обеспечения и инфраструктуры, обсуждая преимущества разных моделей ветвления. По итогам этих встреч я всегда приходила к выводу, что выбор модели зависит от уровня комфорта, размера и характеристик среды команды. В этом разделе рассматриваются некоторые ограничения и проблемы, возникающие при применении обеих моделей ветвления к описанию инфраструктуры.

Проблемы функционально ориентированной разработки

Многие проекты с открытым исходным кодом для разработки приложений и инфраструктуры успешно применяют функционально ориентированную модель. Функционально ориентированная разработка обеспечивает лучшие условия для тестирования и оценки критических изменений независимо друг от друга. Она разделяет изменения между многими участниками и содержит этап ручной проверки перед слиянием с веткой `main`. Системы управления версиями или CI имеют встроенные средства поддержки функционально ориентированной разработки.

Эти преимущества функционально ориентированной модели ветвления в полной мере применимы к IaC. Командам нравится изолировать изменения инфраструктуры в ветке, прежде чем отправлять их в производство. На рис. 7.15 сравниваются изменения правил брандмауэра в среде `TICKET-002` с изменениями лицензии вашего коллеги в среде `TICKET-005`. Вы можете применять изменения в своей ветке, ни с кем не конфликтуя.

Однако функционально ориентированная модель имеет и недостатки. Создание новой среды может потребовать времени и денег (см. главу 12, посвященную управлению затратами). Ваш конвейер может столкнуться с затруднениями при развертывании новых сред, когда несколько членов команды работают над конфигурацией со многими ветками.

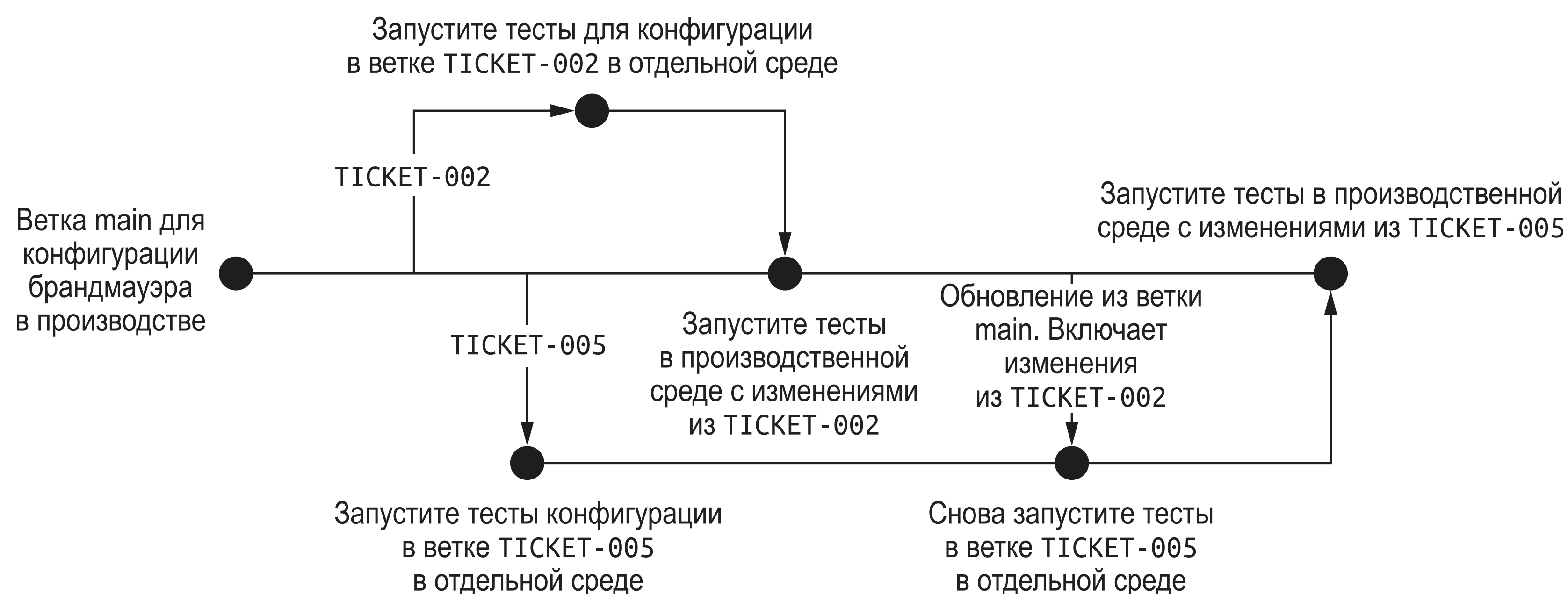


Рис. 7.15 Вы можете создать новую тестовую среду для каждой ветки, чтобы проверять отдельные изменения

Чтобы ускорить создание сред для тестирования, организация может инвестировать в средства выполнения для конвейера, позволяющие параллельно запускать тесты. В качестве альтернативы можно также создать одну постоянную тестовую среду, доступную для всех веток. Однако функционально ориентированная разработка может вызвать конфликты в постоянной тестовой среде, поскольку каждая ветвь применяет изменения асинхронно.

Вместо того чтобы создавать постоянную тестовую среду, вы также можете исключить интеграционные и сквозные тесты из каждой ветки, кроме основной, чтобы оптимизировать затраты. Например, изменение вашего брандмауэра может потребовать только статического анализа, модульных тестов и проверки членами команды перед слиянием с рабочей средой, как показано на рис. 7.16. Вам не нужно создавать уникальную среду тестирования для ветки и объединять ветку с рабочей версией после проверки вручную.

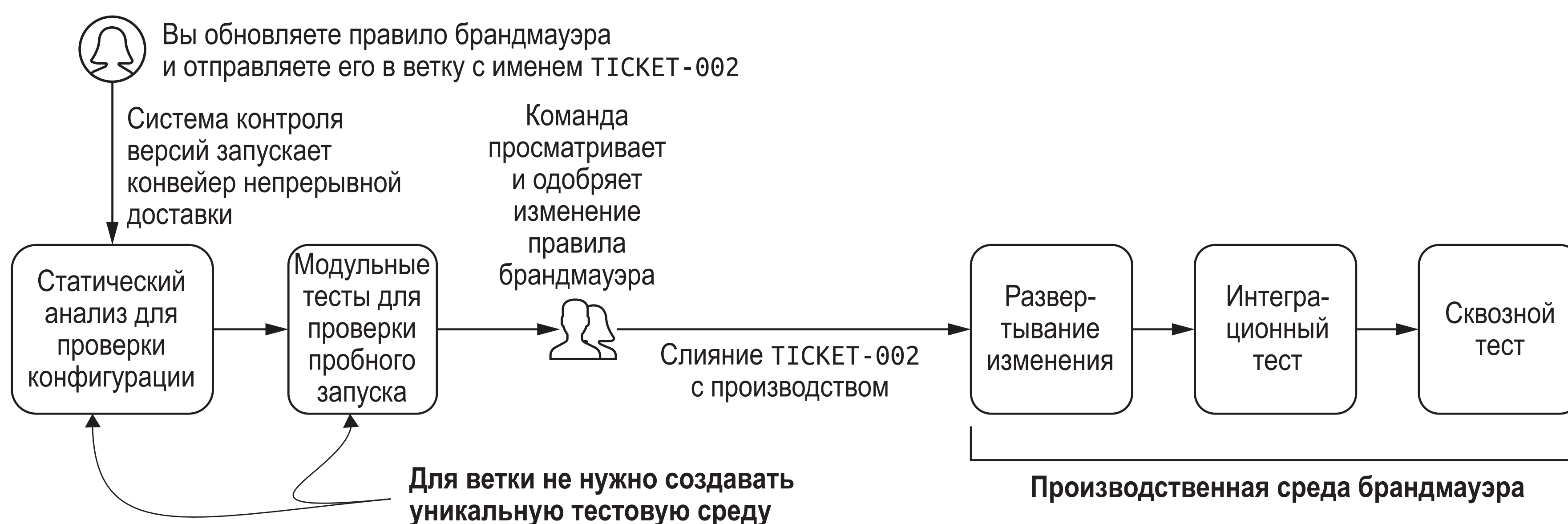


Рис. 7.16 Вы можете исключить интеграцию и сквозные тесты ваших веток, чтобы снизить затраты на использование нескольких сред и параллельную работу конвейеров

Еще одна проблема, с которой вы сталкиваетесь при функционально ориентированной разработке, связана с дисциплиной и уме-

нием пользоваться системой управления версиями. Если вы раньше не использовали контроль версий, вам нужно привыкнуть к процессу ветвления. Ветвление усложняет обратный инжиниринг объединенных изменений и устранение конфликтов.

Например, кто-то может создать ветку, чтобы исправить правило брандмауэра на выходных. Он забывает выполнить слияние с веткой main, а вы не подозреваете, что он поменял настройки брандмауэра. Вы случайно перезаписываете его конфигурацию при обновлении правил брандмауэра! Со временем у вас накопится много веток, и вам придется разбираться, какие из них нужно использовать.

Вы также столкнетесь с проблемой долгоживущих ветвей. Представьте, что ваш коллега уже месяц работает над обновлением лицензии. Он создает новую ветку под названием TICKET-005. Каждые несколько дней ему необходимо проверять наличие обновлений в ветке main и добавлять их в текущее состояние своей ветки.

Однажды наступает момент, когда вам нужно внести изменения, зависящие от обновления лицензии вашего коллеги. Вы начинаете работать над изменениями в ветке под названием TICKET-002, как показано на рис. 7.17. Вы быстро справляетесь со своей работой, но понимаете, что вашему товарищу по команде еще нужно поработать над TICKET-005! Вы ждете еще два месяца, пока ваш товарищ по команде завершит обновление лицензии брандмауэра. Как только он это сделает, вы тратите несколько часов на обновление своей ветки TICKET-002, чтобы наконец можно было развернуть обновления в производстве.

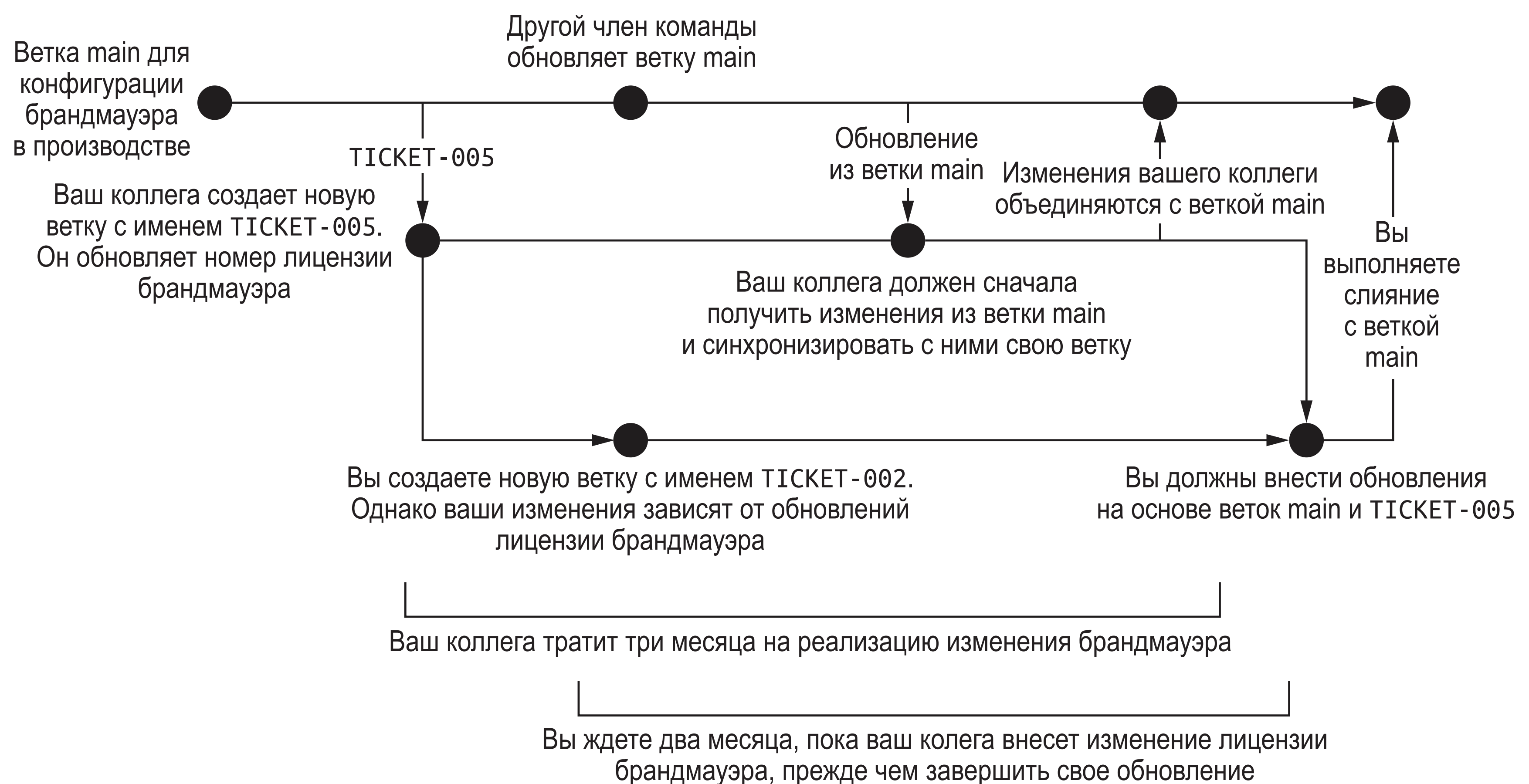


Рис. 7.17 Ветки с длительным сроком существования могут препятствовать развертыванию других изменений в рабочей среде и усложнять процесс ветвления

Функционально ориентированная разработка побуждает вас сохранять ветки в течение длительного времени. Приходится стара-

тельно обновлять долгоживущие ветки, чтобы не отставать от ветки main. В противном случае вы столкнетесь с конфликтами, которые будет сложно разрешить. Иногда единственное решение – удалить заброшенную ветку и перезапустить изменения в новой ветке.

Проблемы магистрально ориентированной разработки

Магистрально ориентированная разработка хорошо согласуется с изменениями инфраструктуры и необходимостью смягчить расхождение конфигурации между средами и состояниями. Вы избегаете сложностей слияния и управления ветками, особенно если вам не хватает опыта работы с Git.

Магистрально ориентированная разработка ориентирована на небольшие, но частые изменения. Вы вносите обновления постепенно, а не тестируете их одним пакетом. В главе 10 я расскажу о приемах постепенного внедрения набора изменений и снижения рисков для инфраструктуры.

У магистрально ориентированной разработки тоже есть несколько недостатков. Прежде чем вносить изменения в производство, необходимо создать специальную тестовую среду. На рис. 7.18 показан идеальный рабочий процесс разработки. После запуска модульных тестов вы развертываете изменение в долгоживущей среде тестирования для выполнения интеграционных и сквозных тестов. Если изменение пройдет тестирование, его проверяют коллеги по коман-

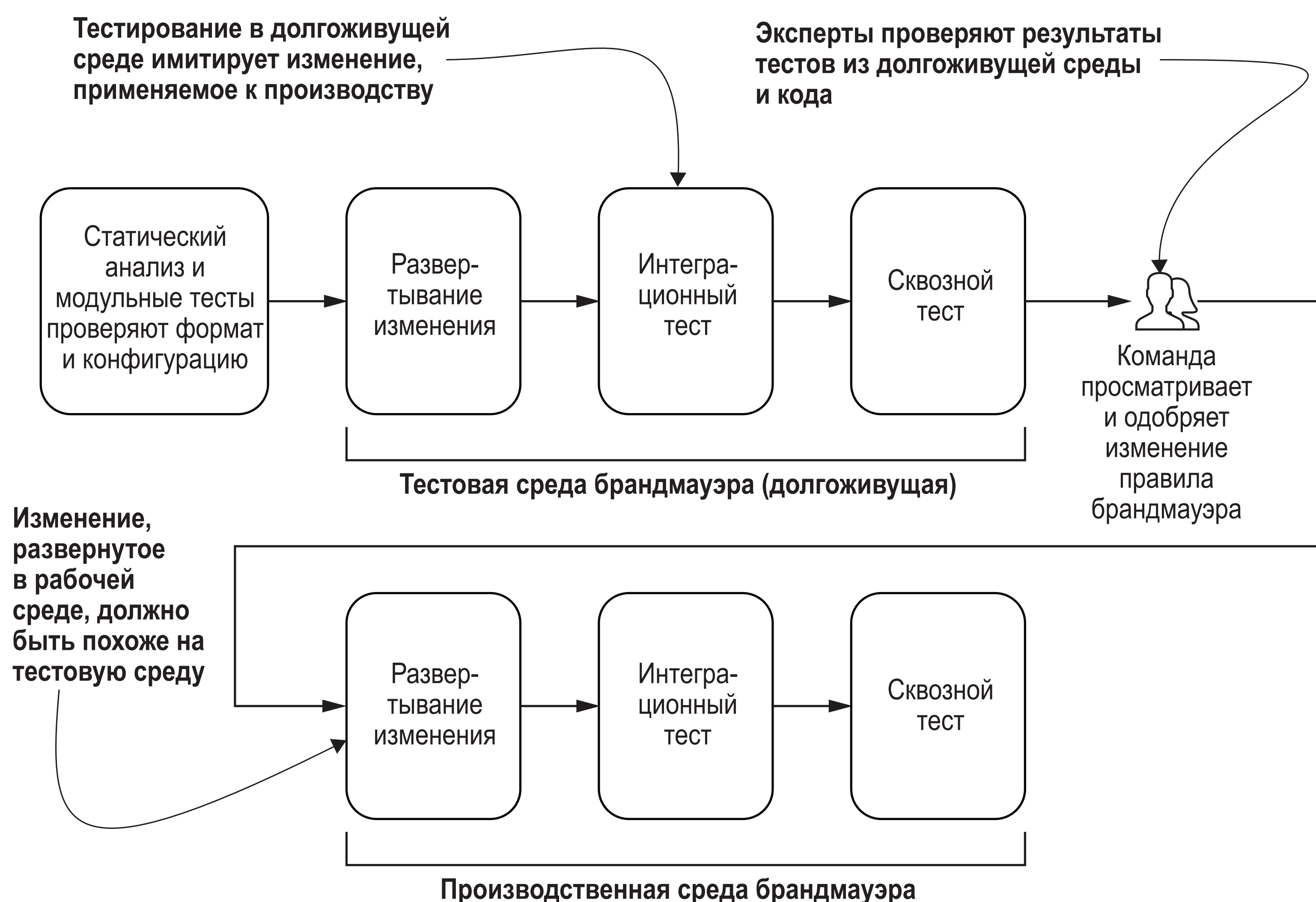


Рис. 7.18 Для магистрально ориентированной разработки требуется специальная тестовая среда, позволяющая имитировать изменения в рабочей среде и повышающая доверие к экспертной оценке

де или эксперты. После одобрения оно поступает в промышленную среду для интеграции и сквозного тестирования.

Вам нужны подробные модульные тесты для форматирования и анализа в соответствии со стандартами вашей команды, а также интеграционные тесты для проверки функциональности. Постоянная среда тестирования увеличивает общую стоимость разработки. Ограниченные и более модульные конфигурации инфраструктуры снижают вероятность конфликтов внутри ресурса или модуля и общую стоимость инфраструктуры для тестов.

Магистрально ориентированная модель плохо согласуется с одобрением изменений вручную. Одобрение может произойти только после того, как кто-то отправит изменения в ветку main. Ваш рецензент должен знать, сработало ли ваше обновление, прежде чем он сможет проверить его синтаксис и конфигурацию. Если вы отправите ошибочную конфигурацию в тестовую среду, вы быстро обнаружите неполадки и отмените обновление, прежде чем его проверит кто-то другой.

В табл. 7.3 суммированы преимущества и ограничения двух моделей разработки. Выбор зависит от типа инфраструктуры, которую настраивает ваша команда, и ее опыта работы с системой контроля версий.

Таблица 7.3. Сравнение функционально ориентированной и магистрально ориентированной моделей разработки

Модель разработки	Преимущества	Недостатки
Функционально ориентированная	Изолирует изменения с помощью веток Изолирует тесты с помощью веток Упорядочивает ручную проверку кода Масштабируется между несколькими командами и соавторами	Требует усердия и знания принципа обновления веток Способствует появлению долгоживущих веток Увеличивает затраты денег и времени
Магистрально ориентированная	Обеспечивает лучшее представление поведения изменений Использует один рабочий процесс управления версиями для всех изменений Поощряет постепенные изменения инфраструктуры для уменьшения потенциального ущерба	Требует долговременной среды тестирования Не содержит этап ручного обзора Требует дисциплины и организации для масштабирования между несколькими командами и соавторами

Вы должны согласовать выбор модели ветвления внутри своей команды. Соглашение о модели помогает обеспечить воспроизводимость изменений и общую доступность системы. Остерегайтесь ограничений каждого подхода и всегда придерживайтесь политики мелких изменений. Независимо от того, какую модель примет ваша команда, вносите изменения в промышленную среду как можно чаще, чтобы уменьшить потенциальный ущерб в случае неудачного обновления.

7.3 Внешняя проверка

На протяжении всей этой главы и ранее в главе 5 я подчеркивала важность включения этапа проверки кода в конвейеры поставки и изменения модулей. Почему вам стоит потратить время на проверку IaC вашего товарища по команде? На что следует обращать внимание при просмотре?

Внешняя проверка позволяет вашим коллегам изучить конфигурацию вашей инфраструктуры на предмет рекомендаций, стандартов и форматирования.

ОПРЕДЕЛЕНИЕ *Внешняя проверка* (peer review) – это практика проверки конфигурации вашей инфраструктуры коллегами по команде или другими командами на предмет соблюдения стандартов и выдачи рекомендаций по улучшению.

В роли проверяющего я обычно сосредотачиваюсь на том, будет ли конфигурация масштабироваться в зависимости от численности команд, останется ли она безопасной или повлияет на зависимости инфраструктуры более высокого уровня. Строгий подход к проверке иногда препятствует внесению изменений в промышленную среду. Но, с другой стороны, коллективную внешнюю проверку можно рассматривать как хороший способ группового обучения стандартизированным практикам и новым моделям. Возможно, вам и вашей команде будет полезно потратить время на обсуждение достоинств и недостатков проекта или реализации.

Чтобы понять важность и недостатки внешней проверки, представьте, что новой команде нужен доступ для чтения к проекту GCP. Вы обновляете код правил управления доступом (листинг 7.1), чтобы читать список пользователей из объекта JSON. Новый код проходит все тесты, и вы ждете несколько дней, пока ваш товарищ по команде проверит изменения.

Листинг 7.1 Первая реализация добавления новой команды в проект GCP

```
import json

GCP_PROJECT_USERS = [                                ①
    (
        'operations',
        'group:team-operations@example.com',
        'roles/editor'
    ),
    (
        'inventory',                                ②
        'group:inventory@example.com',              ②
        'roles/viewer'                              ②
    )
]
```



```

]

class GCPProjectUsers: ❸
    def __init__(self, project, users):
        self._project = project
        self._users = users
        self.resources = self._build() ❹

    def _build(self): ❹
        resources = []
        for user, member, role in self._users: ❺
            resources.append({ ❺
                'google_project_iam_member': [{ ❺
                    user: [{ ❺
                        'role': role, ❺
                        'member': member, ❺
                        'project': self._project ❺
                    }] ❺
                }] ❺
            }) ❺
        return {
            'resource': resources
        }

if __name__ == "__main__":
    with open('main.tf.json', 'w') as outfile: ❻
        json.dump(GCPProjectUsers( ❻
            'infrastructure-as-code-book', ❻
            GCP_PROJECT_USERS).resources, outfile, ❻
            sort_keys=True, indent=2) ❼

```

- ❶ Определяем список пользователей и групп для добавления в проект GCP.
- ❷ Добавляем в проект новую группу `inventory`, имеющую доступ только для чтения.
- ❸ Создаем модуль для пользователей проекта GCP, который использует фабричный паттерн для прикрепления пользователей к ролям.
- ❹ Применяем модуль для создания конфигурации JSON для списка пользователей, добавляемых к ролям GCP.
- ❺ Для каждой группы в списке создается участник проекта Google IAM, которому назначена роль пользователя. Этот ресурс добавляет пользователя к ролям в GCP.
- ❻ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.
- ❼ Когда вы записываете файл JSON, который будет выполнять Terraform, используйте отступ в два пробела.

Эквиваленты AWS и Azure

Чтобы преобразовать листинг кода в AWS, вы должны сопоставить ссылки на проект GCP с учетной записью AWS. Пользователи проекта GCP согласовываются с пользователем AWS IAM. Аналогичным образом вы создаете подписку Azure и добавляете учетную запись пользователя в Azure Active Directory.

Спустя три дня ваш коллега по команде возвращается со следующим отзывом:

- конфигурацию инфраструктуры JSON необходимо сделать с отступом в четыре пробела;
- вам необходимо переименовать группу в team-inventory@example.com;
- вместо определения роли для группы необходимо добавить группу `inventory` в список пользователей с ролью `viewer`.

Ваш коллега объясняет, что первые два пункта относятся к соблюдению командных стандартов. Последнее требование соответствует стандарту безопасности механизма контроля доступа (в соответствии с ним мы связываем список пользователей с ролью вместо назначения роли каждому пользователю). Вы уже задержали обновление на три дня, ожидая экспертной оценки! Теперь вам нужно устранить замечания и подождать еще несколько дней, чтобы получить одобрение.

В главе 6 говорилось о том, что с помощью тестов мы пытаемся получить «неизвестные известные» в виде разрозненных знаний. В данном случае ваш товарищ по команде располагает знаниями, которых у вас не было. Вы решаете добавить несколько модульных тестов, чтобы легче было запомнить и соблюдать стандарты команды.

Новый код в листинге 7.2 содержит модульные тесты (правила линтинга) для проверки конфигурации и стандартов безопасности вашей команды. Один тест проверяет наличие четырех пробелов в JSON. Другой проверяет, соответствуют ли все группы стандарту именования. Последний тест проверяет, используете ли вы правильный ресурс для привязки пользователей к ролям.

Листинг 7.2 Добавление модульных тестов для линтинга стандартов командной разработки

```
import pytest
from main import GCP_PROJECT_USERS, GCPProjectUsers

GROUP_CONFIGURATION_FILE = 'main.tf.json'

@pytest.fixture
def json():
    with open(GROUP_CONFIGURATION_FILE, 'r') as f:
        return f.readlines()

@pytest.fixture
def users():
    return GCP_PROJECT_USERS

@pytest.fixture
def binding():
    return GCPProjectUsers(
        'testing',
```



```

[('test', 'test', 'roles/test')]).resources['resource'][0]

def test_json_configuration_for_indentation(json):
    assert len(json[1]) - len(json[1].lstrip()) == 4, \
        "output JSON with indent of 4"

def test_user_configuration_for_standard_team_name(users):
    for _, member, _ in GCP_PROJECT_USERS:
        assert member.startswith('team-'), \
            "group should always start with `team-`"

def test_authoritative_project_iam_binding(binding):
    assert 'google_project_iam_binding' in binding.keys(), \
        "use `google_project_iam_binding` to add team members to roles"

```

- ❶ Импортируем список пользователей и ролей GCP.
- ❷ Используем Python для чтения файла конфигурации Terraform JSON. В тесте этот блок применяется для проверки наличия в JSON отступа в четыре пробела.
- ❸ Импортируем список пользователей и ролей GCP, включая группу `inventory`, в качестве дополнения к тесту. Тест проверяет, имеет ли каждый пользователь префикс «team-», позволяющий идентифицировать его как группу.
- ❹ Используем тестовый стенд для создания образца пользователя проекта GCP с применением модуля фабрики.
- ❺ Используем Python для чтения файла конфигурации Terraform JSON. В тесте этот блок применяется для проверки наличия в JSON отступа в четыре пробела.
- ❻ Проверяем, что модуль фабрики использует правильный ресурс Terraform для привязки проекта Google IAM. При этом применяется добавление членов команды к определенной роли.

Эквиваленты AWS и Azure

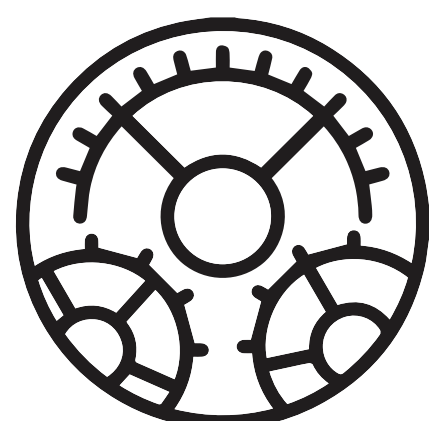
Привязка IAM проекта GCP аналогична ресурсу Terraform `aws_iam_policy_attachment` (<http://mng.bz/5QW7>). Механизм привязки/вложения принудительно отзывает всех пользователей, не определенных как часть ресурса Terraform. На момент публикации книги модель управления доступом Azure использует подход аддитивной политики и не имеет явного способа определения вложений ролей или привязок.

В дальнейшем вы сможете исправлять свои ошибки до экспертной оценки и сокращать цикл обратной связи благодаря автоматическому анализу и модульным тестам. Вашему коллеге не придется придирается к форматированию и стандартам. Однако вы с коллегой вступаете в спор, следует ли вам добавить пользователя к роли или роль к пользователю. Вы решаете вынести этот архитектурный вопрос на обсуждение в более широком составе.

На рис. 7.19 показано, что эффективная экспертная оценка опирается на подход, сочетающий автоматическое тестирование с обсуждением архитектуры в более широком кругу специалистов. Благода-

ря автоматизированным тестам, экспертной оценке и совместному обсуждению в конечном итоге вы получаете безопасную, отказоустойчивую и масштабируемую конфигурацию IaC.

Эффективная групповая оценка для IaC требует сочетания автоматизированных и ручных процессов



Напишите автоматизированные тесты для линтинга и форматирования конфигурации



Просмотрите конфигурацию, чтобы определить зависимую инфраструктуру и изменения в архитектуре*



Используйте парное программирование, чтобы обсуждать изменения и выявлять проблемы на ранней стадии

* Помните о том, сколько времени может занять ручная проверка. Чем больше изменение, тем сложнее проверка и тем больше времени она займет

Рис. 7.19 Автоматизация некоторых проверок и поддержание осведомленности о любых процессах ручной проверки помогают ускорить экспертную оценку

Однако автоматические тесты и экспертные проверки на начальном этапе не решают всех проблем. Можно потратить много усилий на разработку и проверку обновления и все равно получить отрицательную оценку на более поздних этапах. Чтобы выявить архитектурные проблемы на ранних этапах процесса написания IaC, можно организовать совместное программирование с коллегой по команде. В этом методе, называемом парным программированием, участвуют два разработчика.

ОПРЕДЕЛЕНИЕ *Парное программирование* (pair programming) – это совместная работа двух программистов на одной рабочей станции над одной задачей.

Один инженер может уловить нюансы, которые не замечает другой, и наоборот. Парное программирование сопряжено со многими проблемами, включая ограничения ресурсов и личностные конфликты. Большинство компаний не практикуют парное программирование, потому что изначально оно замедляет темпы реализации и влияет на производительность команды. Некоторым людям оно не нравится, потому что их напарник может работать в другом темпе. Парное программирование требует самосознания и дисциплины.

Тем не менее попробуйте применить парное программирование к разработке IaC. С инфраструктурой часто связывают особую терминологию и институциональные знания. Например, нынешние и будущие члены команды должны понимать, почему кто-то выбрал привязку пользователей к роли для контроля доступа к проекту. Парное программирование облегчает обмен знаниями и позволяет

анализировать изменения во время разработки. Со временем ваша команда приобретет опыт быстрого внесения изменений в инфраструктуру без необходимости проверки изменений вручную.

ПРИМЕЧАНИЕ Коллегиальная проверка и парное программирование должны формировать для каждого члена команды безопасное пространство, позволяющее научиться программировать с учетом лучших практик. Детальное описание этих процессов выходит за рамки книги. Для получения дополнительной информации о проверке кода я рекомендую изучить методы разработки Google по адресу <http://mng.bz/6XNR>. Подробнее о парном программировании говорится по адресу <http://mng.bz/o2GD>. Существуют различные приемы организации парного программирования, такие как переключение клавиатур каждые 30 минут и обмен ролями в паре «водитель–навигатор».

Изменение инфраструктуры может повлиять на доступность критически важных для бизнеса систем. Накопление большого пакета изменений может усугубить сбой, затруднив поиск и устранение основной причины. Если вы можете сократить процесс коллегиальной проверки за счет сочетания парного программирования и автоматизации тестирования, вам будет легче сосредоточиться на анализе архитектуры и влиянии изменений инфраструктуры.

7.4 *GitOps*

Что произойдет, если объединить непрерывное развертывание, декларативное конфигурирование, обнаружение расхождений и контроль версий? Все эти методики кажутся совершенно разрозненными, но их совместное использование формирует четкий и эффективный подход к управлению инфраструктурой. Вы объявляете нужную конфигурацию для своей инфраструктуры, добавляете ее в систему контроля версий и развертываете в рабочей среде.

Представьте, что вы хотите обновить платежный сервис с версии 3.0 до 3.2. Платежный сервис работает на Kubernetes и предлагает интерфейс декларативного конфигурирования с использованием DSL. Вы можете передавать файлы YAML для настройки ресурсов в Kubernetes.

На рис. 7.20 показан рабочий процесс, реагирующий на изменения путем сочетания непрерывного развертывания, декларативного конфигурирования и контроля версий. Вы обновляете декларативную конфигурацию до версии 3.2. Контроллер обнаруживает расхождение между текущей конфигурацией и конфигурацией, находящейся в системе контроля версий. Он запускает конвейер поставки

для развертывания новой версии и запуска тестов для проверки ее функциональности.

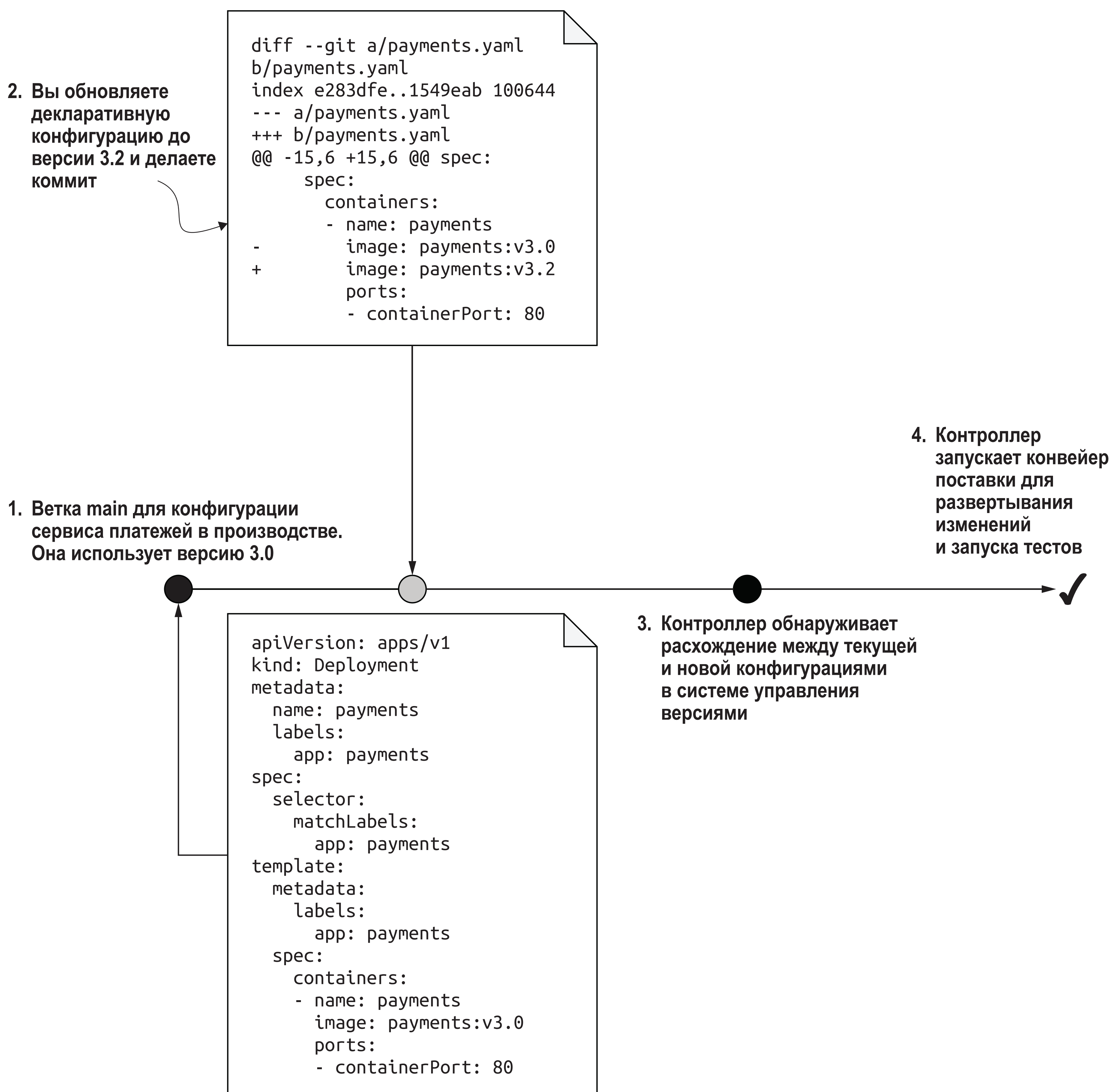


Рис. 7.20 Реализация GitOps может следовать принципам функционально ориентированной модели: создавать pull-request, тестировать изменения в отдельной ветке, добавлять рецензентов и объединять изменения

Почему контроллер постоянно обнаруживает и применяет изменения? Контроллер уменьшает расхождение между ожидаемой конфигурацией и фактическим состоянием. Это гарантирует, что ваша система будет постоянно обновляться.

От этого рабочего процесса у вас может возникнуть ощущение дежавю. Ведь он сочетает в себе все практики по написанию IaC, тестированию и поставке. В этой книге представлена весьма строгая позиция в отношении IaC, точно соответствующая концепции GitOps. В соответствии с этой концепцией команды управляют изменения-

ми инфраструктуры с помощью системы контроля версий, вносят декларативные изменения через IaC и непрерывно развертывают обновления инфраструктуры.

ОПРЕДЕЛЕНИЕ *GitOps* – это подход, который использует декларативную форму IaC для управления изменениями инфраструктуры посредством контроля версий и непрерывного развертывания их в рабочей среде.

GitOps чаще всего ассоциируется с экосистемой Kubernetes. Однако GitOps опирается на парадигму самостоятельного масштабирования практик IaC во всей организации. Вы больше не вносите изменения, заполняя заявки с описанием своих потребностей.

Вместо этого любой компетентный сотрудник организации может выделить новую ветку IaC и сделать коммит изменений. Механизм непрерывного развертывания уменьшает расхождения, поддерживает обновление инфраструктуры и всегда запускает тесты. С помощью запроса на включение и истории коммитов вы можете отслеживать, кто запросил и внес изменения.

ПРИМЕЧАНИЕ Для более близкого знакомства с GitOps и Kubernetes я рекомендую книгу *GitOps and Kubernetes* (Billy Yuen et al., Manning, 2021). Общие сведения о применении GitOps можно найти на <https://opengitops.dev>.

Краткое содержание главы

- Поставка изменений инфраструктуры в промышленную среду обычно включает в себя процесс внешней (консультативной) проверки, в ходе которой вручную проверяются архитектура и влияние изменений.
- Непрерывная интеграция предполагает частое внесение изменений в ветку `main` конфигурации инфраструктуры.
- Непрерывная поставка развертывает изменения в среде тестирования для автоматического тестирования и ожидает одобрения, прежде чем отправить их в промышленной среде.
- Непрерывное развертывание напрямую применяет изменения в рабочей среде без этапа одобрения.
- В зависимости от типа изменения и его частоты вы можете использовать непрерывную интеграцию, поставку или развертывание для внедрения обновлений IaC с помощью автоматических тестов.
- Ваша команда может совместно работать над IaC в соответствии с моделями функционально и магистрально ориентированного ветвления.

- При функционально ориентированной разработке создается ветвь для каждого изменения, что позволяет проводить изолированное тестирование, но требуется знание методов контроля версий.
- При магистрально ориентированной разработке все изменения применяют к ветке `main`, которая выявляет конфликты между изменениями, но требует проверки в тестовой среде перед производством.
- Вы можете автоматизировать проверку форматирования и стандартов и вручную проверять конфигурацию на предмет недостатков архитектуры и конфликтов зависимостей.
- Парное программирование помогает выявить конфликты и проблемы, связанные с изменениями, на ранних этапах процесса разработки.
- Функционал GitOps включает в себя контроль версий, декларативную настройку инфраструктуры и непрерывное развертывание, что позволяет любому пользователю автоматизировать изменения инфраструктуры посредством коммита своей версии кода.

Безопасность и комплаенс

Основные темы главы:

- выбор защиты учетных данных и секретов в IaC;
- построение соответствующей требованиям и безопасной инфраструктуры;
- подготовка комплексных тестов безопасности и комплаенса.

В предыдущих главах я упоминала о важности защиты инфраструктуры как кода и проверки ее соответствия требованиям безопасности и нормам вашей организации. Часто соблюдение этих требований откладывают до более поздних стадий процесса разработки. К этому моменту вы, возможно, уже развернули небезопасную конфигурацию или нарушили правила хранения персональных данных!

Например, представьте, что вы работаете в розничной компании uDress. У вашей команды есть шесть месяцев на создание нового клиентского приложения на GCP. Компании необходимо, чтобы оно было доступно к сезону отпусков. Ваша команда очень усердно работает и разрабатывает функционал, достаточный для запуска. Однако за месяц до развертывания и тестирования нового приложения команда *комплаенса* (compliance, соответствие внутренним и внешним требованиям и нормативам) и безопасности проводит аудит и блокирует выпуск в производство.

Теперь у вашей команды появились новые задачи, направленные на устранение проблем безопасности и комплаенса в соответствии с политикой компании. К сожалению, эти исправления задерживают сроки доставки или, в худшем случае, нарушают функциональность. Возможно, вам следовало подумать об этом с самого начала, чтобы спланировать свою работу!

Политика компании должна обеспечивать соответствие системы требованиям безопасности, аудита и внутреннего контроля. Кроме того, ваши команды по безопасности или комплаенсу часто определяют политики в зависимости от отрасли, страны и других факторов.

ОПРЕДЕЛЕНИЕ *Политика* (policy) – это набор правил и стандартов вашей организации, обеспечивающих соответствие требованиям безопасности, отраслевым стандартам или нормам регулятора.

Эта глава научит вас защищать учетные данные и секреты, а также писать тесты для проверки соблюдения политик безопасности и комплаенса. Продумайте эти вопросы, прежде чем приступить к разработке IaC, и вы сможете создать безопасную инфраструктуру и избежать срыва сроков развертывания. Как любил говорить менеджер, с которым я работала: «Мы встраиваем безопасность в инфраструктуру, а не привинчиваем ее позже».

8.1 Управление доступом и секретами

Я уже рассказывала об идее «встраивания» безопасности в IaC в главе 2. IaC использует два набора секретов. Система оперирует учетными данными API для управления инфраструктурой и конфиденциальными переменными, такими как пароли, для передачи ресурсам. Для удобства защиты и ротации вы можете хранить оба типа секретной информации в менеджерах секретов.

Этот раздел посвящен обеспечению безопасности конвейеров поставки IaC. Сценарий IaC выражает ожидаемое состояние инфраструктуры, в которое часто входят пароли администратора, имена пользователей, закрытые ключи и другая конфиденциальная информация. Конвейеры поставки инфраструктуры управляют развертыванием и выпуском инфраструктуры, которой нужна эта информация.

Предположим, вы создаете конвейеры поставки для развертывания инфраструктуры новой системы uDress. Конвейеры используют набор учетных данных провайдера инфраструктуры для создания и обновления ресурсов. Каждый конвейер также считывает пароль базы данных из менеджера секретов и передает его в качестве атрибута для создания базы данных.

Ваша команда безопасности указывает на две проблемы такого подхода. Во-первых, конвейер поставки инфраструктуры использует полные административные учетные данные для настройки GCP. Во-вторых, конвейер поставки вашей команды случайно выводит в логи пароль администратора базы данных!

Ваш конвейер поставки вместо защиты увеличил поверхность атаки (сумму различных точек атаки) вашей системы.

ОПРЕДЕЛЕНИЕ *Поверхность атаки* (attack surface) описывает сумму различных точек атаки, в которых неавторизованный пользователь может скомпрометировать систему.

Посторонние лица могут использовать учетные данные администратора GCP или пароль базы данных для получения информации и компрометации вашей системы. Вам нужно найти решение, позволяющее лучше защитить учетные данные и пароль базы данных. В идеале это решение должно минимизировать поверхность атаки.

8.1.1 *Принцип наименьших привилегий*

Конвейеры доставки IaC имеют точки атаки, которые позволяют неавторизованным пользователям использовать учетные данные с повышенными правами доступа. Например, в соответствии с главой 7 вы создали конвейер, который непрерывно доставляет изменения инфраструктуры в промышленную среду. Конвейеру необходимы определенные разрешения для изменения инфраструктуры в GCP.

Первоначально ваша команда предоставляет конвейеру полные административные учетные данные, чтобы он мог создавать все ресурсы в GCP. Если кто-то получит доступ к этим учетным данным, он сможет создать и обновить что угодно в системе uDress. Посторонние люди могут использовать конвейер вашей команды для запуска моделей машинного обучения или доступа к другим данным о клиентах!

Конвейеру не требуется доступ ко всем ресурсам. Вы решаете обновить учетную запись, чтобы он использовал только минимальный набор разрешений, необходимый для обновления определенных ресурсов. Вы видите, что IaC создает только сетевые ресурсы, ресурсы Google App Engine и Cloud SQL. Вы удаляете из учетных данных доступ с правами администратора и заменяете его доступом на запись к трем ресурсам.

Когда конвейер работает, как показано на рис. 8.1, новая учетная запись имеет достаточные права для обновления трех наборов ресурсов. Она также получает пароль базы данных от диспетчера секретов перед развертыванием обновлений в сети, приложении и базе данных. После развертывания изменений в тестовой среде вы добавляете модульный тест, чтобы убедиться, что учетные данные больше не содержат административного доступа.

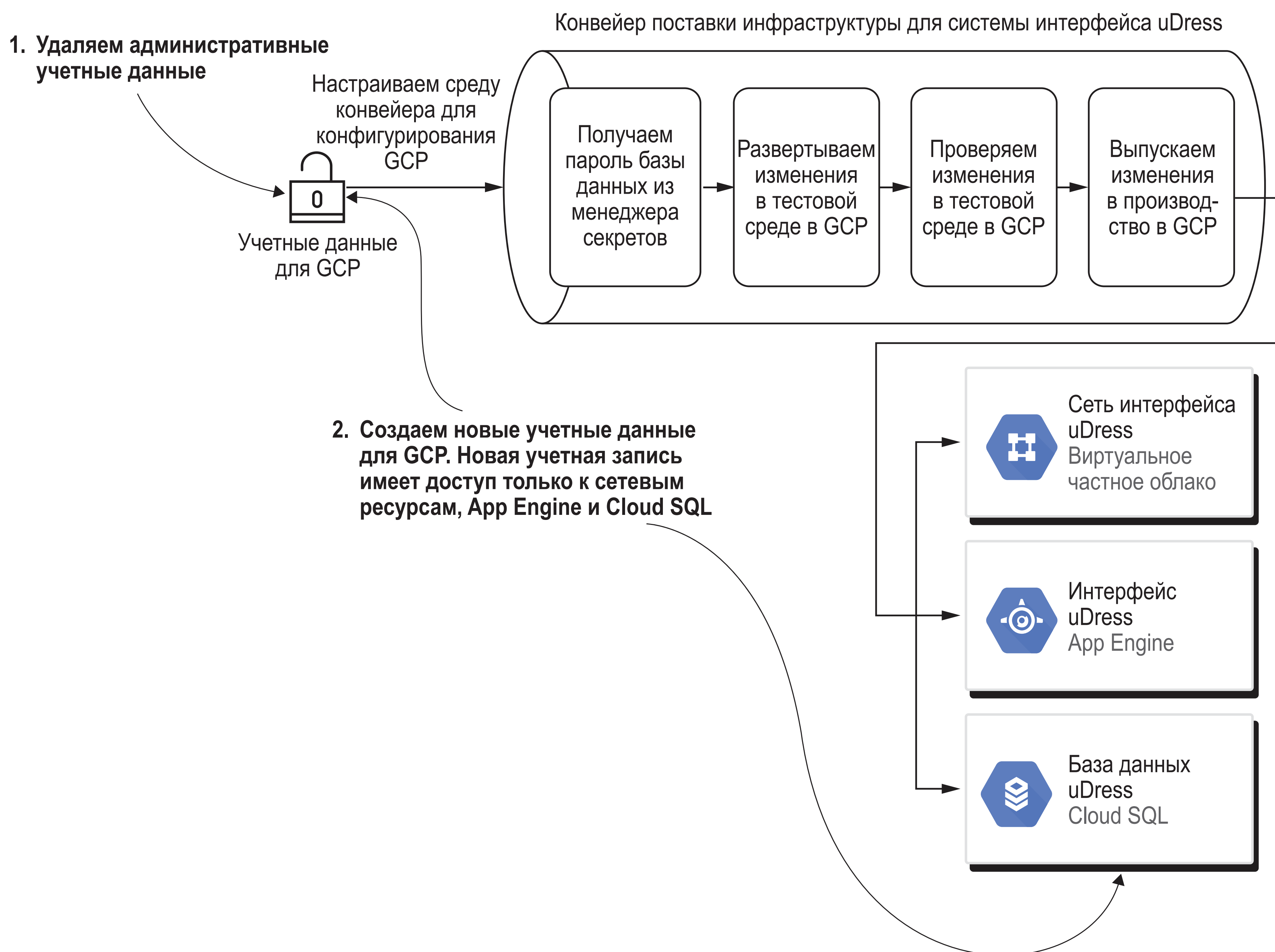


Рис. 8.1 Удалите административные учетные данные из конвейера поставки внешнего интерфейса uDress и ограничьте его доступом к сети, приложению и базе данных

Вы устранили проблему безопасности учетных данных конвейера, используя *принцип наименьших привилегий*. Этот принцип гарантирует, что учетная запись пользователя или службы получит только минимальный доступ, необходимый для выполнения конкретной задачи.

ОПРЕДЕЛЕНИЕ *Принцип наименьших привилегий* (principle of least privilege) означает, что пользователи или учетные записи служб должны иметь минимальные права доступа к системе. Их должно быть достаточно только для выполнения узкого круга конкретных задач.

Соблюдение принципа наименьших привилегий требует времени и усилий. Права доступа обычно настраивают при добавлении новых ресурсов в IaC. Как правило, для этого назначают роли учетной записи конвейера поставки. Группировка прав доступа по ролям соответствует принципу компонентности, поэтому вы можете добавлять и удалять доступ по мере необходимости.

Примените методы работы с модулями из главы 3 для разработки модулей наборов разрешений. Например, вы можете предложить модуль фабрики для веб-приложений uDress, отвечающий за настройку права доступа на запись для конфигурации сети, приложения и базы данных. Любое веб-приложение может применять модуль и корректно воспроизводить необходимый ему минимальный набор привилегий.

Давайте воспользуемся модулем управления доступом, чтобы реализовать управление доступом с наименьшими привилегиями для конвейера поставки внешнего интерфейса uDress, как показано в листинге 8.1. Мы ограничим доступ конвейера учетными данными администратора сети, приложения и Cloud SQL. Эти учетные данные позволяют конвейеру создавать, удалять и обновлять сеть, приложение и базу данных, но запрещают доступ к другим типам ресурсов.

Листинг 8.1 Политика управления доступом с наименьшими привилегиями для внешнего интерфейса

```
import json ❶
import iam

def build_frontend_configuration():
    name = 'frontend'
    roles = [
        'roles/compute.networkAdmin', ❷
        'roles/appengine.appAdmin', ❷
        'roles/cloudsql.admin' ❷
    ]

    frontend = iam.ApplicationFactoryModule(name, roles) ❶
    resources = {
        'resource': frontend._build() ❸
    }
    return resources

if __name__ == "__main__":
    resources = build_frontend_configuration() ❸

    with open('main.tf.json', 'w') as outfile: ❹
        json.dump(resources, outfile, sort_keys=True, indent=4) ❹
```

- ❶ Создаем конфигурацию ролей на основе списка ролей для аккаунта службы, включая сеть, App Engine и Cloud SQL.
- ❷ Импортируем модуль фабрики управления доступом к приложениям для создания ролей внешнего приложения.
- ❸ Применяем метод для создания конфигурации JSON прав доступа к конвейеру.
- ❹ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Google App Engine похож на AWS Elastic Beanstalk или Azure App Service, которые развертывают веб-приложения и сервисы в инфраструктуре, управляемой провайдером.

Google Cloud SQL похож на службу Amazon Relational Database Service (RDS), которая развертывает различные управляемые базы данных. У Azure есть разные службы для конкретных баз данных, например Azure Database для PostgreSQL или Azure SQL Database.

Поскольку вы придерживаетесь принципа наименьших привилегий, будьте осторожны при удалении разрешений. Иногда конвейеру требуются вполне конкретные разрешения для чтения или обновления зависимостей. Вы можете вывести из строя инфраструктуру или приложения, если у конвейера не окажется достаточных привилегий.

Некоторые поставщики инфраструктуры, включая GCP, анализируют разрешения учетной записи службы или пользователя и выдают набор дополнительных разрешений. Вы также можете запустить сторонние инструменты для анализа прав доступа и выявления неиспользуемых разрешений. Я рекомендую использовать эти инструменты для проверки и обновления вашего контроля доступа каждый раз, когда вы добавляете новый ресурс инфраструктуры.

8.1.2 Защита секретов в конфигурации

Помимо использования административных учетных данных из конвейера для доступа к провайдеру инфраструктуры, постороннее лицо может изменить конвейер для кражи конфиденциальной информации об инфраструктуре. Допустим, конвейер поставки внешнего интерфейса выводит в логи пароль администратора базы данных. Любой, кто имеет доступ к логам конвейера, может использовать этот пароль для входа в базу данных!

Чтобы решить проблему безопасности, мы помечаем пароль как *конфиденциальную переменную* (sensitive variable) с помощью инструмента IaC. Инструмент убирает вывод пароля в логи. Мы также устанавливаем в свой инструмент конвейера плагин для обнаружения и редактирования любой конфиденциальной информации, такой как пароль. Мы добавляем эти две конфигурации в свой конвейер, как показано на рис. 8.2, чтобы избежать компрометации пароля базы данных в логах конвейера. В качестве меры предосторожности мы меняем пароль базы данных в диспетчере секретов и делаем это непосредственно в базе данных, а не с помощью IaC.

Дополнительно можно применить инструменты для маскирования пароля в конвейере доставки, скрывая или редактируя информацию, представленную в виде открытого текста.

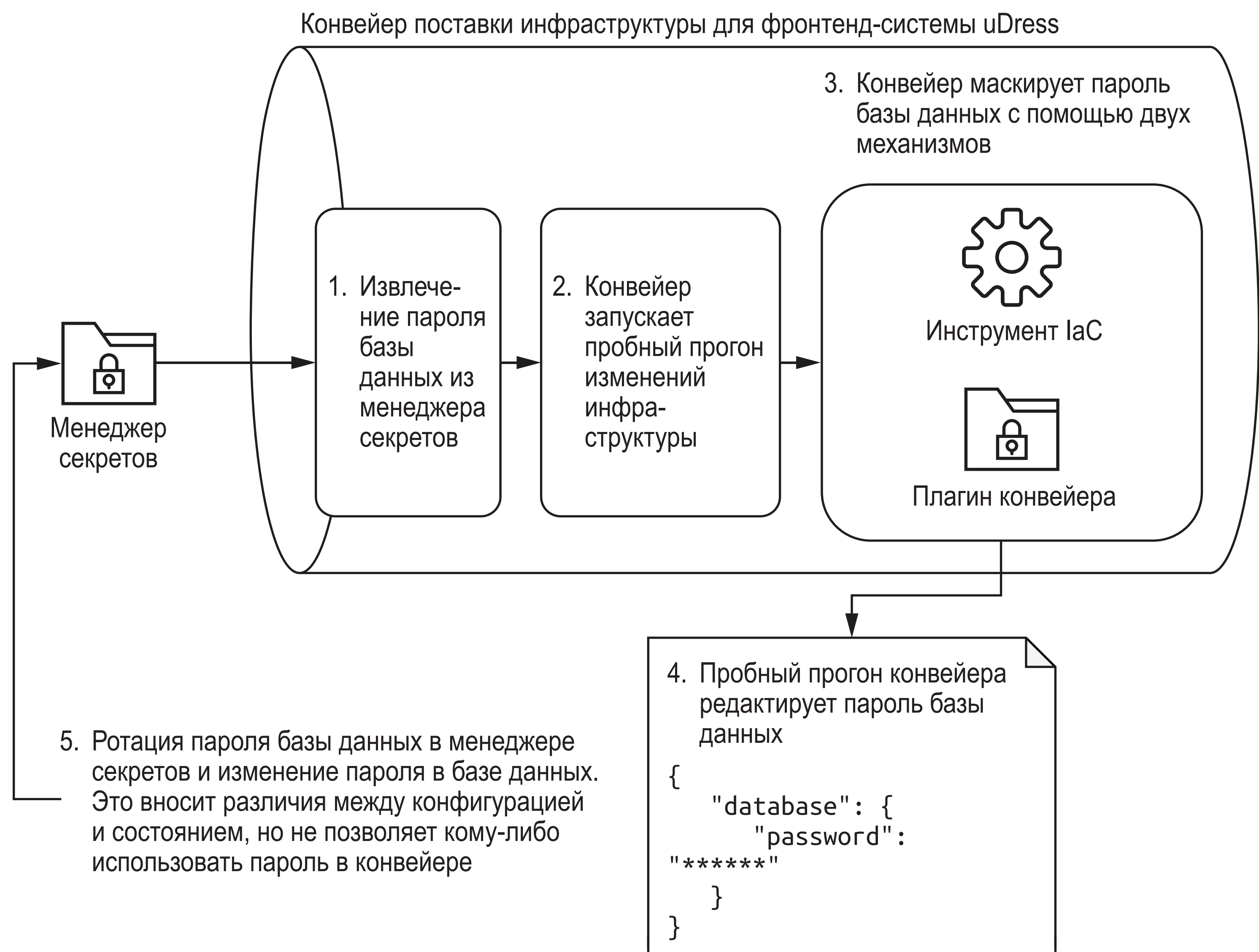


Рис. 8.2 Вы можете защитить административный пароль базы данных, используя инструменты для маскирования значений и ротации учетных записей после применения изменений в IaC

ОПРЕДЕЛЕНИЕ Маскирование вашей конфиденциальной информации означает сокрытие или редактирование открытого текстового представления, чтобы предотвратить кражу информации посторонними лицами.

Использование одного или обоих механизмов предотвратит попадание конфиденциальной информации в логи конвейера. Конфиденциальная информация может содержать пароли, ключи шифрования или идентификаторы инфраструктуры, такие как IP-адреса. Если вы опасаетесь, что кто-то может использовать эту информацию для получения доступа к вашей системе, попробуйте применить маскирование значений в вашем конвейере.

Однако маскирование конфиденциальной информации не гарантирует защиту от несанкционированного доступа. Вам по-прежнему необходим рабочий процесс, позволяющий как можно быстрее сменить раскрытые учетные данные. Одним из решений может служить смена учетных данных с помощью диспетчера секретов, после того как вы используете их для настройки IaC.

Отдельное управление секретами приводит к изменчивости или «изменениям на ходу» в вашем IaC. Несмотря на то что это означает наличие расхождения между фактическим паролем базы данных и паролем, указанным в сценарии конфигурации IaC, изменчивость

пароля не позволяет посторонним людям использовать конвейер IaC и чужие учетные данные.

Создавая IaC, ориентируйтесь на следующий список действий по обеспечению безопасности вашего конвейера поставки, чтобы минимизировать поверхность атаки:

- 1 с самого начала придерживайтесь *политики доступа с наименьшими привилегиями* в учетных данных для провайдера инфраструктуры. Вам необходимо предоставить ровно столько разрешений, сколько нужно для применения и защиты вашего IaC;
- 2 создайте секретные значения с помощью *функции генерации случайной строки* или получите их из менеджера секретов. Избегайте передачи секретов в вашу конфигурацию через статические переменные;
- 3 убедитесь, что ваш конвейер *скрывает конфиденциальные данные* конфигурации при пробном прогоне или выводе команд;
- 4 обеспечьте *механизм быстрого отзыва и ротации* скомпрометированных учетных данных.

Многие обязательные требования к безопасности можно выполнить с помощью менеджера секретов. Менеджер секретов может избавить вас от необходимости статического определения секретов в конфигурации. Хотя некоторые требования относятся к общим стандартам безопасности для конвейеров поставки, они также применимы и к обеспечению безопасности IaC. При необходимости перечитайте главу 2, чтобы освежить в памяти обеспечение безопасности информации с помощью менеджера секретов.

8.2 Инфраструктура тегирования

Итак, начальные требования безопасности инфраструктуры соблюдены, и теперь перед нами стоит задача ее эксплуатации и поддержки. Для эксплуатации инфраструктуры требуется набор паттернов и методов устранения неполадок и аудита. В связи с постоянным расширением и обновлением инфраструктуры нам нужен способ определить назначение и жизненный цикл ресурсов.

Представьте, что мы развернули фронтенд-приложение uDress в производстве. Однако наша команда получает тревожное сообщение от финансового отдела. Сумма в счете нашего провайдера инфраструктуры превысила ожидаемый бюджет за последние два или три месяца. Мы анализируем отчет провайдера, чтобы определить, какие ресурсы внесли наибольший вклад в стоимость услуг. Как узнать владельца и среду каждого ресурса?

GCP допускает использование меток, что позволяет добавлять метаданные к ресурсам для целей идентификации и аудита. Мы обновляем эти метки, включив в них владельца и среду. На рис. 8.3 uDress содержит идентификаторы владельца и среды, стандарты формата

тегов и метаданные автоматизации. Мы разделяем имена и значения тегов дефисом, чтобы теги работали с GCP.

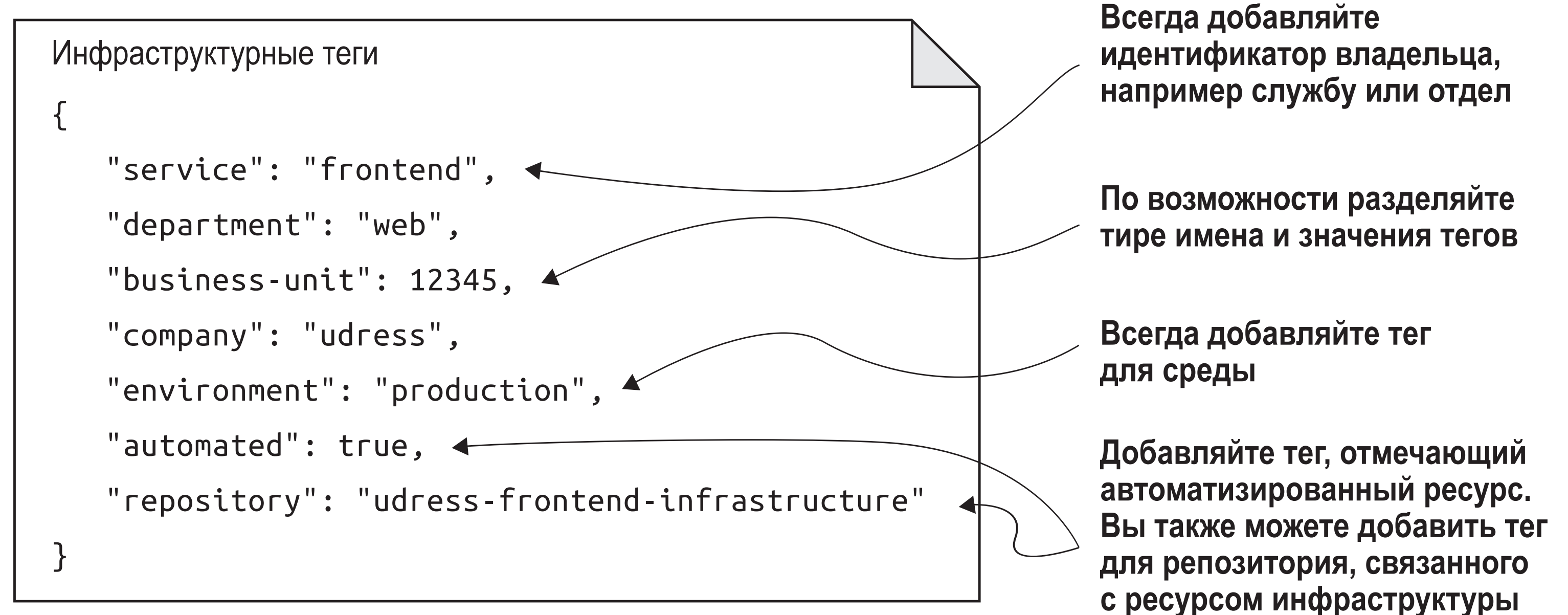


Рис. 8.3 Теги должны содержать идентификаторы владельца, среды и средств автоматизации для облегчения устранения неполадок

Помимо GCP, и другие провайдеры инфраструктуры позволяют добавлять метаданные для идентификации ресурсов. Рекомендую разработать в вашей организации стратегию тегирования, чтобы определить стандартный набор метаданных, используемых для аудита вашей инфраструктурной системы.

ОПРЕДЕЛЕНИЕ *Стратегия тегирования* (tagging strategy) определяет набор метаданных (тегов), используемых для аудита, управления и защиты ресурсов инфраструктуры в вашей организации.

Зачем использовать метаданные в виде тегов? Теги облегчают поиск и проверку ресурсов, а также выполнение действий, необходимых для анализа счетов и соблюдения требований. Вы также можете использовать теги для массовой автоматизации ресурсов инфраструктуры. Массовая автоматизация включает в себя очистку или восстановление обновлений подмножества ресурсов для стабилизации или устранения сбоев системы.

В листинге 8.2 показана реализация стандартных тегов для uDress. В соответствии с главой 3 мы применяем паттерн «прототип», чтобы определить список стандартных тегов для uDress. Мы вызываем модуль тегов uDress, чтобы создать в своем коде список меток для сервера GCP.

Листинг 8.2 Использование модуля Tags для настройки стандартных тегов сервера

```
class TagsPrototypeModule():
    def __init__(
        self, service, department,
```



```

        business_unit, company, team_email,
        environment):
self.resource = {
    'service': service,
    'department': department,
    'business-unit': business_unit,
    'company': company,
    'email': team_email,
    'environment': environment,
    'automated': True,
    'repository': f"${company}-${service}-infrastructure"
}

```

```

class ServerFactory:
    def __init__(self, name, network, zone='us-central1-a', tags={}):
        self.name = name
        self.network = network
        self.zone = zone
        self.tags = TagsPrototypeModule(
            'frontend', 'web', 12345, 'udress',
            'frontend@udress.net', 'production')
        self.resource = self._build()

    def _build(self):
        return {
            'resource': [
                {
                    'google_compute_instance': [
                        {
                            self.name: [
                                {
                                    'allow_stopping_for_update': True,
                                    'boot_disk': [
                                        {
                                            'initialize_params': [
                                                {
                                                    'image': 'ubuntu-1804-lts'
                                                }
                                            ]
                                        }
                                    ],
                                    'machine_type': 'e2-micro',
                                    'name': self.name,
                                    'network_interface': [
                                        {
                                            'network': self.network
                                        }
                                    ],
                                    'zone': self.zone,
                                    'labels': self.tags
                                }
                            ]
                        }
                    ]
                }
            ]
        }

```



```

    }
  ]
}

```

- ❶ Модуль тегов использует паттерн «прототип» для определения стандартного набора тегов.
- ❷ Задаем теги для идентификации владельца, отдела, подразделения для выставления счетов и выделения хранилища ресурса.
- ❸ Передаем необходимые параметры для установки тегов внешнего приложения.
- ❹ Применяем модуль для создания конфигурации JSON сервера.
- ❺ Создаем вычислительный экземпляр (сервер) Google с помощью ресурса Terraform.
- ❻ Добавляем теги из модуля тегов в качестве меток к вычислительному экземпляру Google.

Эквиваленты AWS и Azure

Чтобы перенести код из листинга 8.2 к другому провайдеру облачных услуг, измените ресурс на экземпляр Amazon EC2 или виртуальную машину Azure Linux. Затем передайте `self.tags` в атрибут `tags` для ресурса AWS или Azure.

Как узнать, какие теги добавить? В главе 2 было сказано, что нужно стандартизировать именование и тегирование ресурсов инфраструктуры. Обсудите эти соображения с отделами аудита, безопасности и финансов. Это поможет определить, какие теги вам нужны и как их использовать. Как минимум у меня всегда есть теги для следующих характеристик:

- служба или команда;
- электронная почта команды или другой канал связи;
- среда развертывания (разработка или производство).

Например, предположим, что команда безопасности uDress проверяет ресурсы внешнего интерфейса и обнаруживает неправильно настроенную инфраструктуру. Члены команды могут проанализировать теги, определить сервис и среду, в которых возникла проблема, и обратиться к команде, создавшей ресурс.

Дополнительно можно использовать следующие теги:

- автоматизация – помогает отличить созданные вручную ресурсы от автоматизированных;
- репозиторий – позволяет сопоставлять ресурс с его исходной конфигурацией в системе контроля версий;
- бизнес-подразделение – идентификатор для выставления счетов или возврата платежей в системе бухгалтерского учета;
- соответствие – указывает, соответствует ли ресурс требованиям политики компании или требованиям к обработке персональных данных.

Принимая решение о тегах, убедитесь, что они соответствуют общему набору ограничений, чтобы вы могли применять одни и те же

теги к любому провайдеру инфраструктуры. Большинство провайдеров инфраструктуры имеют ограничения на количество символов в тегах. Обычно я предпочитаю формат, в котором имена тегов и значения записаны строчными буквами и разделены дефисами. Хотя вы можете использовать CamelCase, не все провайдеры поддерживают теги, чувствительные к регистру.

У разных поставщиков инфраструктуры различаются ограничения на максимальную длину тега. Большинство провайдеров поддерживают максимальную длину ключа тега 128 символов и значения тега 256 символов. Вам придется найти компромисс между описательными именами (см. главу 2) и ограничениями длины тегов у провайдера!

Еще одна важная часть вашей стратегии тегирования заключается в принятии решения о том, следует ли *удалять ресурсы, не имеющие тегов*. Подумайте о принудительном применении тегов ко всем ресурсам в своей производственной среде. Тестовая среда может поддерживать ресурсы без тегов для ручного тестирования. Я не рекомендую удалять ресурсы без тщательного изучения последствий. Вы же не хотите случайно удалить важный ресурс.

8.3 Политика как код

Защита доступа и секретов в конвейерах поставки инфраструктуры, а также тегирование ресурсов инфраструктуры несомненно помогают соблюдать требования безопасности и прочих правил. Но при этом сохраняется необходимость выявлять небезопасную или несовместимую конфигурацию инфраструктуры, прежде чем она будет запущена в промышленной среде. Вы должны обнаружить проблему до того, как кто-то другой обнаружит ее в вашей производственной системе.

Представьте себе подключение внешнего приложения uDress к другой базе данных. Вы добавляете правило брандмауэра, разрешающее весь входящий трафик в базу данных для тестирования. После тестирования вы планируете удалить базу данных, поэтому не помечаете ее тегом.

Вы забываете про брандмауэр и настройку тегов и отправляете обновление на проверку. К сожалению, коллега по команде не замечает вашу оплошность при проверке кода и отправляет изменения в промышленную среду. Две недели спустя вы обнаруживаете, что кто-то посторонний получил доступ к некоторым данным! Однако у вас нет тегов для идентификации скомпрометированной базы данных.

Что вы могли бы сделать по-другому? Вспомните важность модульных тестов или статического анализа конфигурации инфраструктуры, о которых говорилось в главе 6. Вы можете применить

те же методы для написания тестов, предназначенных для проверки соблюдения требований безопасности и политики хранения данных.

Вместо того чтобы полагаться на коллегу, который должен обнаружить проблему, вы могли бы выразить политику в виде кода для статического анализа конфигурации на предмет разрешающего правила брандмауэра или отсутствия тегов. Методика «политика как код» заключается в тестировании метаданных инфраструктуры и проверке их на соответствие различным требованиям.

ОПРЕДЕЛЕНИЕ Методика *политика как код* (или *раннее тестирование безопасности, статический анализ IaC*) заключается в тестировании метаданных инфраструктуры и проверке их на соответствие различным требованиям перед внесением изменений в промышленную среду. Политика как код включает в себя правила для инструментов динамического анализа или сканирования уязвимостей.

О долгосрочной выгоде от автоматизации и тестирования IaC говорилось в главах 1 и 6. Написание политики в виде кода тоже требует первоначальных затрат труда и времени. Но зато в дальнейшем будет происходить автоматическая проверка соответствия каждого изменения, которое вы хотите внести в производственную среду. Вы сведете к минимуму неприятные сюрпризы после проверки вашей системы специалистами по комплаенсу и безопасности. Со временем вы уменьшите долгосрочные затраты времени и ускорите развертывание в промышленной среде.

8.3.1 *Механизмы и стандарты политики*

Существующие инструменты помогают реализовать политику в виде кода, оценивая метаданные на основе набора правил. Большинство инструментов тестирования в этой области используют механизм политик. Механизм политики принимает политики как код в качестве входных данных и оценивает ресурсы инфраструктуры на предмет соответствия.

ОПРЕДЕЛЕНИЕ Механизм политики (policy engine) принимает политики как код в качестве входных данных и оценивает метаданные ресурсов на соответствие политикам.

Многие механизмы политики анализируют и проверяют поля в конфигурации или состоянии инфраструктуры. На рис. 8.4 механизм политики извлекает JSON или другие метаданные из состояния IaC либо системы. Затем он передает метаданные на проверку безопасности или политики. Механизм запускает тест для анализа полей, проверки их значений и завершается сбоем, если фактические значения не соответствуют ожидаемым.

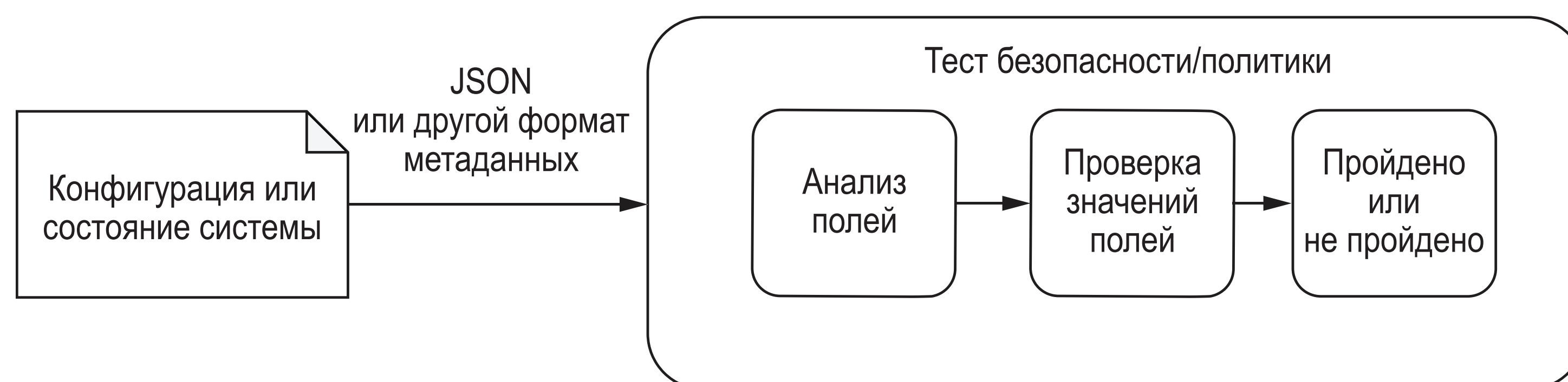


Рис. 8.4 Тесты безопасности и политики анализируют конфигурацию или состояние системы на наличие правильных значений полей и завершаются неудачно, если они не соответствуют ожидаемому значению

Этот рабочий процесс применяется к политикам, инструментам кода и любым тестам, которые вы пишете самостоятельно. Инструменты политики как кода упрощают тестирование, поскольку они абстрагируют сложность анализа полей и проверки значений. Однако готовые инструменты не охватывают все значения или варианты использования, которые вам нужно протестировать.

В результате вам придется разработать свой собственный механизм политики, соответствующий вашим целям. В примерах этой главы я использую `pytest`, среду тестирования Python, в качестве примитивного «механизма политики» для проверки безопасности и совместимости конфигурации.

Механизмы политики

Экосистема политики как кода имеет разные инструменты для разных целей. Большинство инструментов относятся к одному из трех вариантов использования, каждый из которых выполняет совершенно разные функции и существенно различается по поведению:

- 1 тесты безопасности для конкретных платформ;
- 2 тесты на соответствие отраслевым или нормативным стандартам;
- 3 пользовательские политики.

В табл. 8.1 представлен неполный список механизмов политик для инструментов конфигурирования – как проприетарных, так и с открытым исходным кодом. Я кратко обрисовала несколько технологических интеграций и варианты использования для каждого инструмента.

Вам придется смешивать и сочетать инструменты, чтобы охватить все варианты использования. Универсальных инструментов не существует. Некоторые инструменты предлагают функциональность, которую вы можете использовать для создания собственных политик. Но сначала рассмотрите возможность расширения существующего инструмента с помощью настраиваемых политик, чтобы вы могли задавать шаблоны и настройки по умолчанию для своих групп по безопасности, комплаенсу и разработке. На самом деле у вас со временем наберется пять или шесть механизмов политики для охвата необходимых вам инструментов, платформ и политик.

Таблица 8.1. Примеры механизмов политики для инструментов конфигурирования

Инструмент	Примеры применения	Технологические интеграции
AWS CloudFormation Guard	Тестирование безопасности для некоторых платформ Пользовательские политики	AWS CloudFormation
HashiCorp Sentinel	Тестирование безопасности для некоторых платформ Пользовательские политики	HashiCorp Terraform
Pulumi CrossGuard	Тестирование безопасности для некоторых платформ Пользовательские политики	Pulumi SDK
Open Policy Agent (базовая технология для Fugue, Conftest, Kubernetes, Gatekeeper и др.)	Тестирование безопасности для некоторых платформ (зависит от инструмента) Тестирование на соблюдение политики и отраслевых стандартов (зависит от инструмента) Пользовательские политики	Разные (полный список доступен по адресу www.openpolicyagent.org/docs/latest/ecosystem/)
Chef InSpec	Тестирование безопасности для некоторых платформ Пользовательские политики	Разные (полный список см. на маркетплейсе Chef)
Kyverno	Тестирование безопасности для некоторых платформ Пользовательские политики	Kubernetes

Обратите внимание, что я не рассматриваю здесь какие-либо инструменты безопасности или политики, ориентированные на конкретное оборудование центров обработки данных, потому что выбор этого оборудования часто зависит от потребностей вашей организации. Вы также можете найти некоторые открытые проекты, кроме примеров, перечисленных в табл. 8.1. Я часто замечаю, что эти инструменты и их интеграции заменяются более новыми, поскольку экосистема быстро меняется.

Сборка образа и управление конфигурацией

В инструментах сборки образов не так уж много инструментов безопасности или политики, поскольку для них обычно разрабатывают свои тесты. Инструменты управления конфигурацией используют аналогичный подход. Вам нужно будет найти инструменты, разработанные сообществом сторонников открытого кода, или встроенные инструменты, которые проверяют безопасность и конфигурацию политики.

Отраслевые или нормативные стандарты

Анализируя табл. 8.1, вы могли заметить, что лишь немногие инструменты поддерживают проверку политики на соответствие отраслевым или нормативным стандартам. Большинство этих политик существуют в виде документации, и вам часто приходится писать их самостоятельно. Иногда удастся найти готовые наборы тестов политики, которые вам нужно будет дополнить собственными тестами.

Например, Национальный институт стандартов и технологий (NIST) в США публикует список показателей безопасности в рамках Национальной программы контрольных списков (<https://ncp.nist.gov/repository>). Рецензент этой книги также рекомендовал руководства по технической реализации мер безопасности (STIG) Министерства обороны США, включая стандарты технического тестирования и конфигурации.

ПРИМЕЧАНИЕ Да, в этом разделе я не упоминаю многие стандарты и инструменты. Стандарты, на которые я ссылаюсь, применимы к Соединенным Штатам, а не ко всему миру. К тому времени, когда вы будете читать эту книгу, у механизмов политики могут измениться функции, интеграции или статус открытого исходного кода, а отраслевые или нормативные стандарты получают обновленные проекты. Если вы хотите порекомендовать мне новые инструменты, свяжитесь со мной через GitHub по адресу <https://github.com/joatmon08/tdd-infrastructure>.

8.3.2 Тесты безопасности

Что следует протестировать, чтобы защитить свою инфраструктуру? Иногда предлагают взять за основу показатели, отражающие лучшие практики обеспечения безопасности системы. Однако вам может потребоваться написать свою собственную политику как код для конкретных платформ и инфраструктуры вашей компании. Давайте начнем устранять нарушения безопасности вашей базы данных, о которых мы говорили в начале главы. К счастью, тестовые данные не содержат ничего важного. Но вы не хотите, чтобы ваши коллеги по команде копировали и развертывали конфигурацию прямо в промышленной среде. Для предотвращения развертывания тестовой версии IaC в промышленной среде вы пишете тест для защиты сети, в которой расположена база данных.

Базе данных требуется очень строгое правило брандмауэра с наименьшими привилегиями (минимальный доступ). На рис. 8.5 показано, как реализовать тест для получения конфигурации межсетевого экрана из IaC. Конфигурация передается в тест, ко-

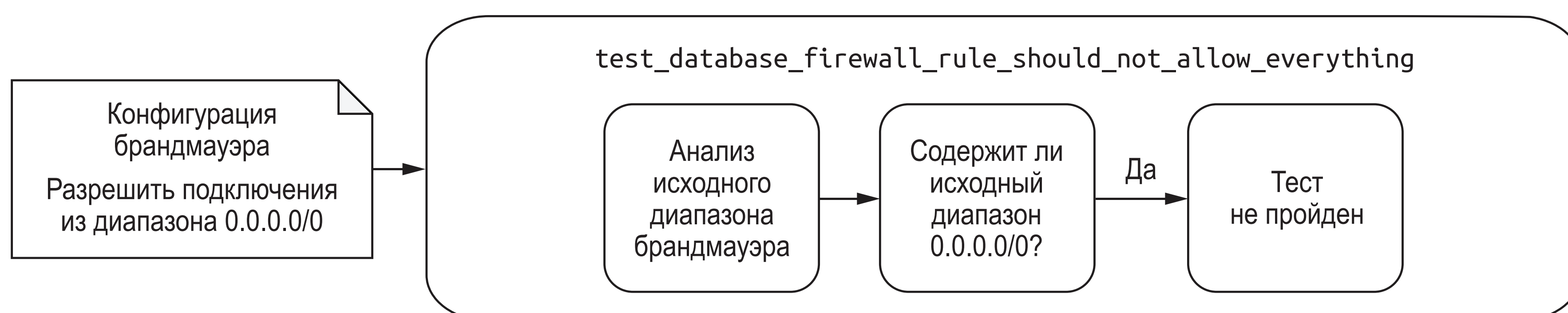


Рис. 8.5 Получаем значение исходного диапазона из конфигурации правила брандмауэра и определяем, не содержит ли оно чрезмерный разрешенный диапазон

торый анализирует исходный диапазон из правила брандмауэра. Если диапазон содержит разрешающее правило 0.0.0.0/0, тест не пройден.

GCP использует диапазон 0.0.0.0/0 для обозначения того, что любой IP-адрес может получить доступ к базе данных. Если кто-то получит доступ к вашей сети, он сможет получить доступ к вашей базе данных, если у него есть имя пользователя и пароль. Ваш новый тест заблокирует развертывание еще до того, как чрезмерно свободное правило, такое как 0.0.0.0/0, будет отправлено в промышленную среду.

В листинге 8.3 реализован тест правила брандмауэра в Python. Показанный здесь код открывает файл конфигурации JSON, извлекает список `source_ranges` и проверяет, содержит ли этот список диапазон 0.0.0.0/0.

Листинг 8.3 Использование теста для обнаружения правила брандмауэра, разрешающего диапазон 0.0.0.0/0

```
import json
import pytest
from main import APP_NAME, CONFIGURATION_FILE

@pytest.fixture(scope="module")
def resources():
    with open(CONFIGURATION_FILE, 'r') as f:
        config = json.load(f)
    return config['resource']

@pytest.fixture
def database_firewall_rule(resources):
    return resources[0][
        'google_compute_firewall'][0][APP_NAME][0]

def test_database_firewall_rule_should_not_allow_everything(
    database_firewall_rule):
    assert '0.0.0.0/0' not in \
        database_firewall_rule['source_ranges'], \
        'database firewall rule must not ' + \
        'allow traffic from 0.0.0.0/0, specify source_ranges ' + \
        'with exact IP address ranges'
```

- ❶ Загружаем конфигурацию инфраструктуры из файла JSON.
- ❷ Анализируем блок ресурсов из файла конфигурации JSON.
- ❸ Анализируем ресурс брандмауэра сервера Google, определенного Terraform из конфигурации JSON.
- ❹ Используем описательное имя теста, объясняющее политику правила брандмауэра, которое не должно разрешать весь трафик.
- ❺ Проверяем, что диапазон 0.0.0.0/0 (разрешить все) не определен в правилах.
- ❻ Используем описательное сообщение об ошибке, описывающее, как исправить правило брандмауэра, например удалить 0.0.0.0/0 из списка диапазонов.

Эквиваленты AWS и Azure

Правило брандмауэра в GCP эквивалентно группе безопасности AWS (<http://mng.bz/Qvvm>) или группе безопасности сети Azure (<http://mng.bz/XZZY>). Чтобы перенести код, создайте ресурс группы безопасности у выбранного вами облачного провайдера. Затем отредактируйте тест, чтобы переключить `source_ranges` GCP на атрибут `security_rule.source_port_range` для Azure или атрибут `ingress.cidr_blocks` для AWS.

Представьте, что ваш новый коллега хочет провести несколько тестов базы данных со своего ноутбука. Он вносит изменения, чтобы открыть правило брандмауэра для `0.0.0.0/0` в IaC. Конвейер запускает код Python для генерации JSON:

```
$ python main.py
```

Конвейер запускает модульные тесты, сверяя файл JSON с конфигурацией. Тест обнаруживает, что правило брандмауэра содержит диапазон `0.0.0.0/0` в списке разрешенных диапазонов, и выдает ошибку:

```
$ pytest test_security.py
===== short test summary info =====
FAILED
test_security.py::test_database_firewall_rule_should_not_allow_everything -
    ➔AssertionError: database firewall rule must not allow traffic
    ➔from 0.0.0.0/0, specify source_ranges with exact IP address ranges
===== 1 failed in 0.04s =====
```

Ваш коллега читает описание ошибки и понимает, что правило брандмауэра не должно разрешать весь трафик. Он может исправить свою конфигурацию, добавив IP-адрес своего ноутбука в список разрешенных диапазонов.

Как и функциональные тесты в главе 6, тесты безопасности знакомят остальную часть вашей команды с идеальными методами обеспечения безопасности инфраструктуры. Хотя тесты не обязательно выявляют все нарушения безопасности, они сообщают важную информацию об ожиданиях безопасности. Превращение «неизвестных известных» рекомендаций по безопасности в «известные известные» исключает повторную ошибку.

Эти тесты также помогают масштабировать методы обеспечения безопасности в вашей организации. Ваш товарищ по команде чувствует свою ответственность за исправление конфигурации. Кроме того, ваша служба безопасности проводит меньше расследований и последующих действий в случае нарушений безопасности. Если безопасность станет частью общей ответственности, это сократит время и усилия, необходимые для исправления проблем в будущем.

Положительное и отрицательное тестирование

В примере с диапазоном IP-адресов базы данных мы проверили, что диапазон IP-адресов *не* является полностью открытым (0.0.0.0/0). Этот процесс, называемый *отрицательным тестированием*, убеждает, что фактическое значение *не* совпадает с нежелательным. Вы также можете выполнить *положительное тестирование*, чтобы убедиться, что атрибуты совпадают с ожидаемым значением.

В некоторых публикациях предлагают придерживаться какого-то одного типа для всех тестов безопасности или политики. Однако я обычно пишу тесты как с положительными, так и с отрицательными проверками. Такая комбинация лучше отражает цель требований безопасности и политики. Например, вы можете использовать отрицательный тест для проверки любого диапазона IP-адресов на соответствие любой конфигурации инфраструктуры, написанной любой командой. С другой стороны, если у вас есть диапазон IP-адресов, который должен входить в каждое правило брандмауэра, например VPN-соединение, вы можете использовать положительный тест.

Вы можете написать тесты для проверки других аспектов безопасности конфигураций, включая следующие:

- порты, диапазоны IP-адресов или протоколы в других сетевых политиках;
- контроль доступа, исключающий доступ от имени администратора к ресурсам инфраструктуры, серверам или контейнерам;
- конфигурация метаданных для предотвращения использования метаданных экземпляра;
- конфигурация журнала доступа и аудита для информации безопасности и управления событиями (SIEM), например для балансировщиков нагрузки, IAM или сегментов хранения;
- версии пакета или конфигурации для устранения уязвимостей.

Этот неполный список охватывает некоторые наиболее распространенные конфигурации. Проконсультируйтесь со своей командой по безопасности или другими отраслевыми экспертами для получения дополнительной информации и идей для тестов.

8.3.3 Тесты политики

Тесты безопасности должны удостовериться, что вы свели к минимуму поверхность атаки из-за неправильной конфигурации IaC. Однако вам потребуются другие тесты для аудита, отчетности, выставления счетов и устранения неполадок. Например, ваша тестовая база данных должна иметь метку, чтобы кто-то мог идентифицировать ее владельца и сообщить о нарушении безопасности.

Команда комплаенса uDress напоминает о необходимости добавить теги в вашу базу данных GCP, чтобы они могли идентифици-

ровать владельца базы данных. Они также уведомляют вас о том, что нарушение безопасности привело к масштабированию ресурсов базы данных, что увеличило затраты на облачные вычисления. Без тегов команде комплаенса было трудно определить, к кому обращаться по поводу проблем с безопасностью и увеличения расходов.

Мы добавляем теги в конфигурацию базы данных. Чтобы не забыть о тегах в будущем, мы применяем рабочий процесс, показанный на рис. 8.6, для реализации модульного теста проверки тегов. Как и в случае с проверкой безопасности правил брандмауэра, мы анализируем файл JSON с конфигурацией базы данных, чтобы проверить, правильно ли мы заполнили поля тегами. Если обнаруживаются пустые поля, тест не пройден.

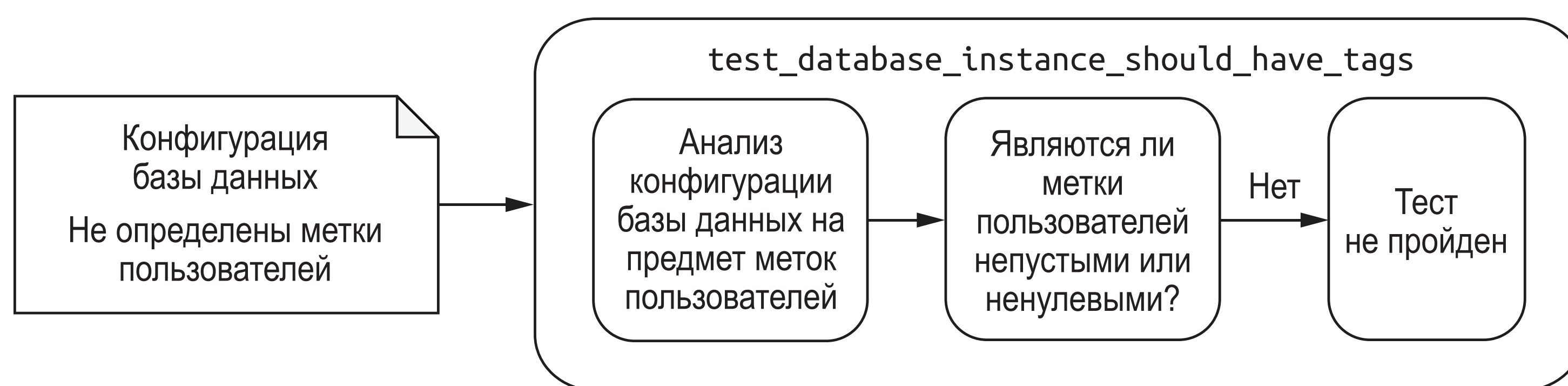


Рис. 8.6 Структура теста, который анализирует конфигурацию базы данных и проверяет список тегов в метках ее пользователей

Тест политики ведет себя аналогично тесту безопасности. Однако он проверяет теги, а не диапазоны источников IP-адресов. Хотя эта политика не укрепляет защиту инфраструктуры, она облегчает устранение неполадок и различение ресурсов.

Давайте реализуем тестовый рабочий процесс. В следующем листинге представлен тест, проверяющий наличие тегов в параметре `user_labels` GCP.

Листинг 8.4 Использование теста для анализа конфигурации базы данных на наличие тегов

```

import json
import pytest
from main import APP_NAME, CONFIGURATION_FILE

@pytest.fixture(scope="module")
def resources():
    with open(CONFIGURATION_FILE, 'r') as f:
        config = json.load(f)
    return config['resource']

@pytest.fixture
def database_instance(resources):
    return resources[2][
        'google_sql_database_instance'][0][APP_NAME][0]

def test_database_instance_should_have_tags(database_instance):
  
```



```

assert database_instance['settings'][0]['user_labels'] \
    is not None
assert len(
    database_instance['settings'][0]['user_labels']) > 0, \
    'database instance must have `user_labels`' + \
    'configuration with tags'

```

5
5
6
6
6
6

- ❶ Загружаем конфигурацию инфраструктуры из файла JSON.
- ❷ Анализируем блок ресурсов из файла конфигурации JSON.
- ❸ Анализируем метки пользователей в экземпляре базы данных Google SQL, определенном Terraform, на основе конфигураций JSON.
- ❹ Используем описательное имя теста, объясняющее политику маркировки базы данных.
- ❺ Проверяем, что метки пользователей в базе данных не имеют пустого списка или нулевого значения.
- ❻ Используем описательное сообщение об ошибке, напоминающее о необходимости добавления тегов к меткам пользователей GCP.

Эквиваленты AWS и Azure

Чтобы перенести код листинга 8.4 в AWS или Azure, измените экземпляр базы данных Google SQL на объект PostgreSQL из любого облака. Вы можете использовать AWS RDS (<http://mng.bz/yvvJ>) или базу данных Azure для PostgreSQL (<http://mng.bz/M552>). Затем проанализируйте ресурс экземпляра базы данных на предмет атрибута tags. И Azure, и AWS используют теги.

Мы добавляем тест в процедуру проверки безопасности. В следующий раз, когда кто-то внесет изменения и забудет теги, тест провалится. Автор обновления прочитает сообщение об ошибке и исправит конфигурацию IaC, добавив теги. Аналогичным образом можно реализовать тесты для других политик организации, включая следующие:

- обязательные теги для всех ресурсов;
- количество ответственных лиц, одобряющих изменение;
- географическое расположение инфраструктурного ресурса;
- логи и целевые ресурсы для аудита;
- отделение данных разработки от данных промышленной среды.

Этот неполный список охватывает некоторые общие конфигурации. Вам следует проконсультироваться со своей командой комплаенса или ознакомиться с отраслевыми нормативами для получения дополнительной информации и идей для тестов. При написании тестов обязательно включайте в них четкие сообщения об ошибках с указанием того, какие политики проверяет тест.

8.3.4 Паттерны и методики

По мере написания новых тестов безопасности и политик растет уверенность в том, что ваша конфигурация остается безопасной

и соответствующей требованиям. Как передать свой опыт команде и компании? Здесь пригодятся некоторые методики и паттерны тестирования для проверки безопасности и соответствия инфраструктуры. Далее я более подробно расскажу о методах и паттернах написания тестов безопасности и политик.

Используйте подробные названия тестов и сообщения об ошибках

В предыдущих примерах вы могли обратить внимание на подробные названия и развернутые сообщения об ошибках в тестах безопасности и политики компании uDress. Эти имена и сообщения кажутся многословными, но они сообщают товарищам по команде, что именно предусматривает политика и как им следует ее исправить! В главе 2 я представила метод проверки качества именования и кода. Попросите кого-нибудь взглянуть на исходный код теста. Если он поймет предмет и цель теста, то сможет обновить свою конфигурацию, чтобы она соответствовала подходу политики как кода.

Разбивайте тесты на модули

Некоторые *паттерны модулей* из главы 3 можно применить к подходу «политика как код». Допустим, команда обработки платежей uDress хочет позаимствовать наши тесты безопасности и политики для своей инфраструктуры. Мы делим политики базы данных на тесты базы данных `database-tests`, а политики брандмауэра на тесты брандмауэра `firewall-tests`.

Команда безопасности также просит вас добавить бенчмарк Центра интернет-безопасности (Center for Internet Security, CIS). Этот отраслевой бенчмарк включает в себя тесты для проверки соблюдения лучших методик безопасной настройки GCP. После добавления бенчмарка безопасности вы понимаете, что у вас слишком много тестов, чтобы их можно было отслеживать в нескольких репозиториях.

На рис. 8.7 все эти тесты перенесены в репозиторий с именем `gcp-security-test`. Репозиторий организует все тесты для инфраструктуры GCP uDress. Команды внешнего интерфейса и платежей uDress могут обращаться к общедоступному репозиторию, импортировать тесты и запускать их в своей конфигурации. Тем временем команда безопасности может обновлять тесты безопасности в одном месте в репозитории `gcp-security-tests`.

Как и в случае с инфраструктурным подходом к структурам репозитория кода, вы можете разместить политику вашей организации в виде кода в одном репозитории или разделить ее на несколько репозиториев в зависимости от среды. В любом случае убедитесь, что все команды имеют представление о тестах безопасности и политик, чтобы организация могла организованно развертывать совместимую инфраструктуру.

Кроме того, разделите тесты по бизнес-подразделениям, функциям, средам, поставщикам инфраструктуры, ресурсам инфраструктуры, стеку или бенчмаркам. Вы должны развивать тесты разного типа индивидуально по мере изменения вашего бизнеса. Одним бизнес-подразделениям может потребоваться тот или иной тип тестов, а другим – нет. Поэтому имеет смысл разделить тесты и выполнять их выборочно по мере необходимости.

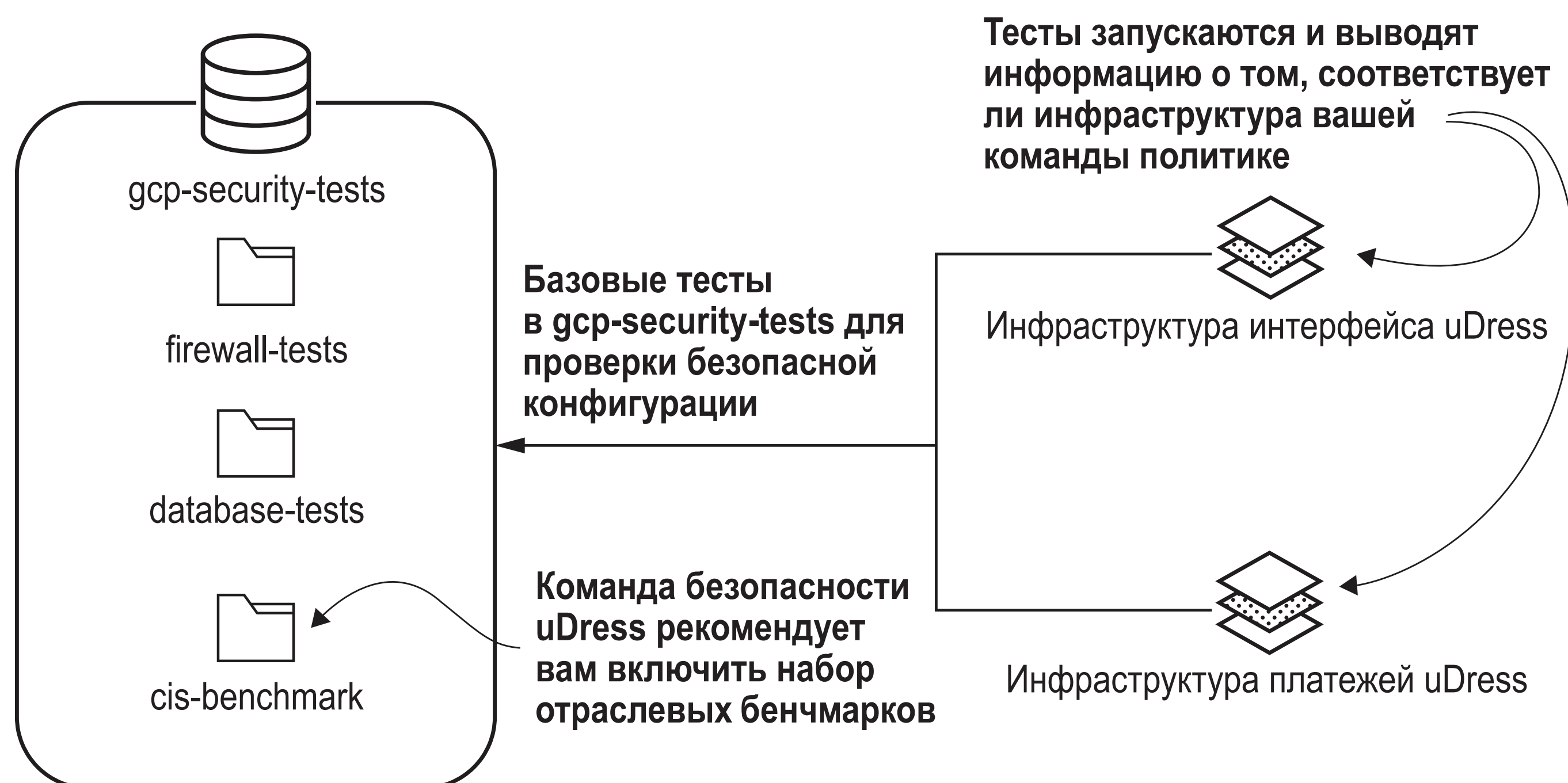


Рис. 8.7 Добавление политики как кода в общедоступный репозиторий для распространения среди всех команд, создающих инфраструктуру

Добавьте политику как код в конвейеры поставки

Допустим, ваша команда решила обязательно выполнить тесты безопасности и политики перед выпуском в рабочую среду, поэтому добавила их как *этап своего конвейера поставки*. Проверка политики как кода выполняется *после* развертывания изменений в тестовой среде, но *перед* выпуском в производство. Вы получаете быструю обратную связь об изменениях в инфраструктуре, расставляете приоритеты по функциональности, но проверяете политику перед началом эксплуатации.

Команда безопасности также добавляет тестирование политики как кода в промышленной среде. Этот динамический анализ постоянно проверяет безопасность системы и отслеживает любые внезапные или аварийные изменения в инфраструктуре.

На рис. 8.8 показан рабочий процесс статического анализа в конвейере поставки и динамического анализа работающей инфраструктуры для проверки изменений конфигурации и упреждающего решения проблем с ресурсами. После развертывания изменений в тестовой среде мы запускаем тесты безопасности и политики. Они должны пройти до выпуска обновлений в промышленной среде. После обновления ресурсов мы проверяем инфраструктуру с помощью аналогичных тестов безопасности и политики в рабочем режиме.

Тесты для статического и динамического анализов могут различаться. Некоторые тесты, такие как проверка доступа к конечной

точке в реальном времени, применимы только к производственной инфраструктуре. В результате приходится выполнять несколько тестов до и после запуска в производство.



Рис. 8.8 Тесты безопасности и политики проверяют неправильную конфигурацию инфраструктуры и предотвращают попадание неудачных обновлений в производство

Если тесты статического анализа занимают слишком много времени, вы можете запустить подмножество тестов после того, как внесете изменение в производственную среду. Однако вам придется быстро устранять любые нарушения безопасности или комплаенса. Поэтому я рекомендую выполнять наиболее важные тесты безопасности и политики как часть вашего конвейера.

Построение образа

Вы можете столкнуться с необходимостью построения неизменяемых образов сервера или контейнера. Записав нужные пакеты в образ сервера или контейнера, вы можете создавать новые серверы с обновлениями без проблем, связанных с обновлениями на месте.

Используйте уже знакомый рабочий процесс инфраструктурного конвейера с политикой в виде кода для создания неизменяемых образов. Рабочий процесс включает в себя модульные тесты для проверки сценариев

на соответствие конкретным требованиям установки, например реестру пакетов компании, а также интеграционные тесты на тестовом сервере для проверки соответствия версий пакетов политике и безопасности.

Вы всегда можете использовать динамический анализ в виде агента для сканирования сервера и проверки его конфигурации на соответствие правилам. Например, Sysdig предлагает Falco, инструмент проверки безопасности во время работы, который запускается на действующем сервере и проверяет соответствие правилам.

Большинству команд не нужно, чтобы их тесты безопасности или политики блокировали передачу всех обновлений в производство. Например, что, если клиентам потребуется доступ к общедоступным конечным точкам инфраструктуры? Тест для проверки только частной сети может оказаться неприменимым. Иногда вам придется делать исключения в своей политике безопасности.

Определите уровни обязательного исполнения

По мере создания новых политик в виде кода вы должны определить наиболее важные из них и сделать исключения для других. Например, команда безопасности uDress определила, что политика тегирования базы данных является строго обязательной. Конвейер поставки должен быть остановлен, если кто-то забыл указать теги, и для устранения блокировки развертывания эти теги необходимо добавить.

Мы определяем три категории политик, как показано на рис. 8.9. Команда безопасности требует, чтобы мы исправили теги базы данных перед отправкой в рабочую среду. Однако команда делает исключение для правила брандмауэра, поскольку клиентам необходим доступ к конечной точке. Команда также дает несколько советов по более безопасным конфигурациям инфраструктуры.



Рис. 8.9 Разделение политик на три категории применения, от которых зависят особенности развертывания обновлений перед отправкой в производство

Я классифицирую политики по трем категориям обязательности (терминология заимствована из HashiCorp Sentinel):

- *строгое требование* соблюдения обязательных политик;

- *мягкое требование* соблюдения политик, которые могут потребовать ручного анализа исключений;
- *консультации* по обмену знаниями о передовом опыте.

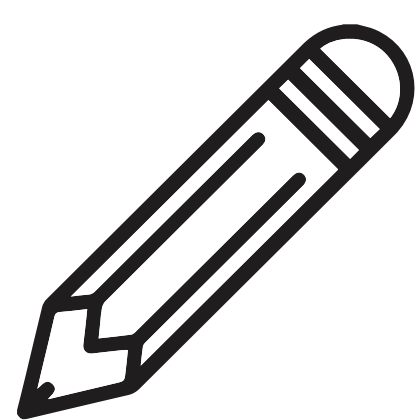
Группа безопасности относит правило брандмауэра к мягким требованиям. Некоторые общедоступные балансировщики нагрузки должны разрешать доступ с адреса 0.0.0.0/0. Если проверка правила брандмауэра не пройдена, представитель команды безопасности должен просмотреть правило и вручную утвердить изменение в конвейере.

Команда безопасности рассматривает стандарты CIS как рекомендации по обмену знаниями и передовыми методиками и просит исправить конфигурацию, если это возможно, но не требует строгого соблюдения перед производством.

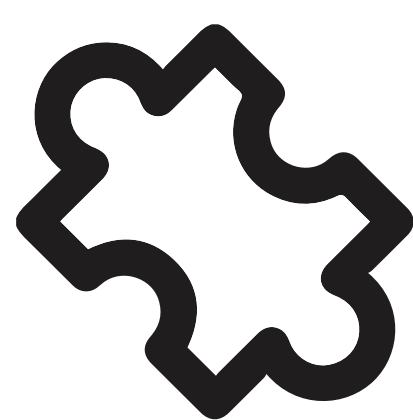
Должны ли мы запускать тесты безопасности, перед тем как изменения попадут в производство? В конце концов, они отнимают драгоценное время! Если вы беспокоитесь, что тесты безопасности и политики будут слишком долго задерживать обновления, запустите *строго обязательные* тесты перед развертыванием в производстве.

Вы можете запускать необязательные или рекомендованные тесты асинхронно, чтобы только строгие тесты блокировали ваш конвейер. Но я *не* рекомендую запускать *все* тесты безопасности и политики асинхронно, поскольку вы можете временно внести в рабочую среду несовместимую конфигурацию, даже если быстро исправите ее после запуска асинхронных тестов!

На рис. 8.10 обобщены паттерны и методы тестирования, такие как написание подробных названий тестов и сообщений об ошибках. Как и в случае с инфраструктурой, вы можете разбивать тесты на модули на основе функций и добавлять тесты в конвейеры поставки в производство.



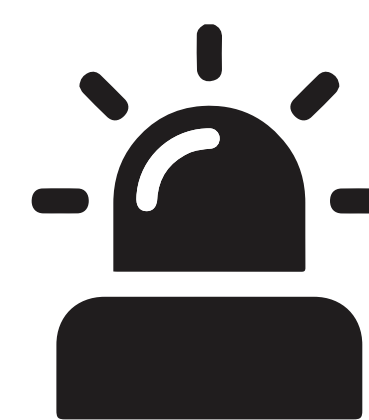
Напишите подробные имена и сообщения об ошибках, чтобы донести требования безопасности и политики до коллег



Разделите тесты безопасности и политики по бизнес-подразделениям, функциям, средам, провайдерам инфраструктуры или стекам



Добавьте тесты безопасности и политики в конвейеры поставки в качестве шлюза перед производством



Определите уровни обязательности, чтобы обозначить обязательные или рекомендательные политики

Рис. 8.10 Тесты безопасности и политики проверяют конфигурацию инфраструктуры и предотвращают попадание неправильных обновлений в производство

Независимо от инструмента, бенчмарка безопасности или правил политики, вам следует представлять и передавать рабочие методы в форме тестов. Следование этим паттернам и практикам поможет вам и вашим товарищам по команде улучшить свои знания в области безопасности и комплаенса.

По мере своего роста организация вводит новые политики, а вы расширяете перечень тестов. Чем раньше вы начнете применять методику политики как кода, тем легче будет внедрять практики обеспечения безопасности и комплаенса в ваш IaC. Если вы не можете найти готовый инструмент для запуска необходимых вам тестов, попробуйте разработать собственные тесты для анализа IaC.

Краткое содержание главы

- Соблюдение политики компании гарантирует, что ее системы соответствуют требованиям безопасности, аудита и комплаенса. Ваша компания определяет политику, исходя из отраслевой принадлежности, законов страны и других факторов.
- Согласно принципу минимальных привилегий, мы предоставляем пользователю или учетной записи службы только тот минимальный доступ, который им необходим.
- Убедитесь, что ваша конфигурация IaC использует учетные данные с наименьшими привилегиями доступа к провайдеру инфраструктуры. Наименьшие привилегии не позволяют посторонним использовать учетные данные и создавать неавторизованные ресурсы в вашей среде.
- Используйте инструмент для удаления или редактирования конфиденциальной информации в конвейере поставки.
- Меняйте имена пользователей и пароли, созданные IaC, после их применения в конвейере.
- Тегирование инфраструктуры с указанием службы, владельца, электронной почты, учетной информации, среды и сведений об автоматизации упрощает идентификацию, аудит безопасности и выставление счетов.
- В соответствии с подходом политики как кода мы тестируем отдельные метаданные инфраструктуры и проверяем их соответствие безопасной конфигурации и критериям комплаенса.
- Используйте подход политики как кода для проверки безопасности вашей инфраструктуры перед внесением изменений в промышленную среду, но после функционального тестирования вашей системы.
- Применяйте простые и понятные паттерны IaC и модулей для управления и масштабирования политики в виде кода.
- Отнесите каждую проверку безопасности и политики к одной из трех категорий применения, таких как строго обязательная (необходимо исправить), желательная (проверка вручную, попытка исправить) и рекомендательная (рекомендуется соблюдать, но развертывание в производстве не блокируется).

Часть III

Управление сложностью производственной инфраструктуры

Если в организации хотя бы несколько команд используют технологию IaC и взаимодействуют между собой, неизбежно встает вопрос о том, как управлять сложностью производственной инфраструктуры и вносить в нее изменения. В этой заключительной части описываются методы обновления инфраструктуры посредством IaC и изменения самой стратегии IaC. Вы научитесь изменять свою инфраструктуру и совершенствовать инструменты и конфигурации!

В главе 9 вы узнаете, как вносить изменения в инфраструктуру и минимизировать вероятность неудачных изменений. В главе 10 описан процесс рефакторинга IaC, который может вам понадобиться после расширения системы. Если рефакторинг приводит к неполадкам в инфраструктуре, вы можете использовать общие методы, описанные в главе 11, для отката изменений и устранения проблем.

По мере дальнейшего развития вашей системы вы столкнетесь с проблемами стоимости. В главе 12 представлены рекомендации по методам управления затратами на инфраструктуру. Наконец, глава 13 завершает книгу описанием методов использования инструментов и модулей с открытым исходным кодом, а также замены или обновления существующих инструментов.

Развертывание обновлений

Основные темы главы:

- принятие решений о том, когда использовать такие подходы, как blue-green развертывание, для обновления инфраструктуры;
- обеспечение неизменности для создания сред в нескольких регионах с помощью IaC;
- выбор стратегий для обновления инфраструктуры, хранящей данные (базы данных).

Предыдущие главы были посвящены полезным практикам и паттернам для реализации модульности, разделения зависимостей, тестирования и развертывания изменений инфраструктуры. Однако вам также необходимо научиться управлять изменениями в инфраструктуре, построенной по принципу IaC. В этой главе вы узнаете, как планировать стратегию изменения в IaC, ориентированных на неизменяемость, для минимизации последствий потенциального сбоя.

Давайте вернемся к компании Datacenter for Veggies из главы 5 и ее попыткам разбить IaC на модули. Компания приобретает стартап Datacenter for Carnivorous Plants¹ в качестве дочерней компании.

¹ Центр обработки данных о плотоядных растениях. – Прим. перев.

Такие плотоядные растения, как венерина мухоловка, требуют особых условий выращивания.

Поэтому Datacenter for Carnivorous Plants нуждается в глобальной сети и специфических серверах и компонентах. Раньше большинство команд сами настраивали свои IaC, но теперь понимают, что их код не может справиться с такими масштабными изменениями. Они обращаются за помощью к вам – инженеру компании Datacenter for Veggies, который имеет богатый опыт работы с IaC.

Команда инженеров направит вас к команде Cape Sundew (капская росянка), чтобы сначала обновить ее инфраструктуру. Капская росянка – выносливое хищное растение, поэтому лучше других переносит сбои системы, вызывающие колебания температуры и полива. Все ресурсы инфраструктуры команды Cape Sundew существуют в одном репозитории с несколькими файлами конфигурации.

Вы исследуете и рисуете архитектуру системы для выращивания росянки. На рис. 9.1 показано правило региональной балансировки (балансировщик нагрузки), которое отправляет трафик в региональную сеть с кластером контейнеров и тремя серверами. Весь трафик циркулирует в одном регионе, а не глобально.

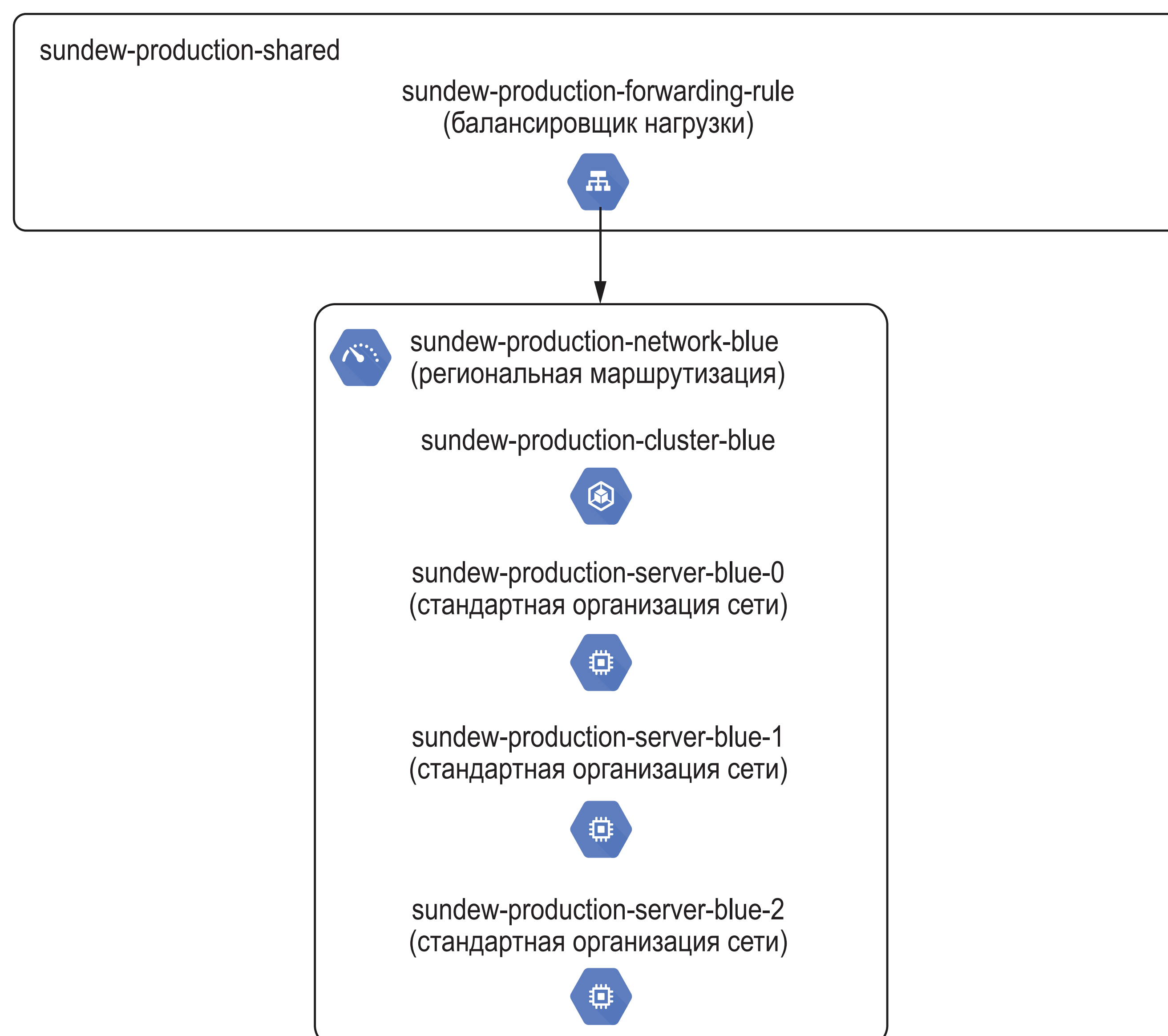


Рис. 9.1 Приложения Cape Sundew используют инфраструктурную систему с общими ресурсами балансировки нагрузки, тремя серверами и одним кластером контейнеров

В связи со слиянием компаний вам нужно изменить все ресурсы для глобальной балансировки трафика, а не маршрутизации в од-

ном регионе. Однако команда Cape Sundew определила все эти ресурсы в одной конфигурации IaC, которую мы называем паттерном «одиночка» (см. главу 3). Ресурсы также используют общее состояние инфраструктуры.

Как внедрить изменения сетевых настроек и минимизировать риск возникновения сбоев? Вас беспокоит возможное нарушение работы приложения полива, если изменить инфраструктуру, внося обновления на ходу. Например, перенастройка сети на использование глобальной маршрутизации может повлиять на серверы, обслуживающие приложение для полива. Вы помните из главы 2, что вы можете использовать идею неизменяемости для создания новой инфраструктуры с новыми изменениями. Если вы сможете применить эти методы к системе, то сможете изолировать и протестировать изменения в новой среде, не затрагивая старую. В этой главе вы узнаете, как изолировать и вносить изменения в IaC.

ПРИМЕЧАНИЕ Демонстрация стратегий внесения изменений нуждается в достаточно большом и сложном примере. Если вы запустите полный пример, вы понесете расходы, превышающие уровень бесплатного пользования GCP. В эту книгу включены только самые необходимые строки кода, а остальные опущены для удобства чтения. Полный код можно найти в репозитории книги по адресу <https://github.com/joatmon08/manning-book/tree/main/ch09>. Если вы преобразуете эти примеры для AWS и Azure, вы также понесете расходы. По возможности я предлагаю инструкции по переносу примеров на платформу облачного провайдера по вашему выбору.

9.1 Подготовка к изменениям

Вы сразу переходите к изменению системы по выращиванию росянки. К сожалению, вы случайно удалили атрибут конфигурации, который помечает сервер тегом `blue` (синий), что разрешает трафик между всеми экземплярами `blue` в сети. Вы отправляете изменения в конвейер поставки, чтобы протестировать конфигурацию и применить ее в рабочей среде.

Тестирование не замечает упущенный тег. К счастью, ваша система мониторинга отправляет вам предупреждение о том, что приложение для полива не может связаться с вашим новым сервером! Вы перенаправляете все запросы на дублирующий экземпляр сервера, чтобы росянки получали полив во время отладки.

Вы понимаете, что не стоит начинать с изменения системы. Система по выращиванию росянки имеет свою архитектуру и инструменты, с которыми необходимо разобраться перед началом

работы. Вам также необходимо знать, есть ли в системе резервные копии или альтернативные среды, готовые к использованию, если вы что-то сломаете. Что вам следует сделать, прежде чем вносить изменения?

9.1.1 Наличие четкого плана действий

Вы всегда рискуете сделать ошибки и создать другие проблемы при изменении IaC. Вам нужны тестирование, мониторинг и *наблюдаемость системы* (возможность определить внутреннее состояние системы по ее выходным данным), чтобы гарантировать, что вы не нарушили работу системы во время внесения изменений. Если вашей системе не хватает прозрачности, вы не сможете быстро устранять проблемы, связанные с неработающими изменениями.

Прежде чем менять систему для выращивания росянки, вы решаете кое-что проверить в своей системе. На рис. 9.2 показаны ваши действия. Сначала вы добавляете тест для выявления отсутствующих тегов. Затем вы анализируете свою систему мониторинга на наличие проверок и показателей работоспособности как системы, так и приложений. Наконец, вы создаете дубликаты серверов в качестве резерва на тот случай, если вы сломаете существующие серверы. Вы можете отправить трафик на резервный сервер, если случайно нарушите работу промышленного сервера.

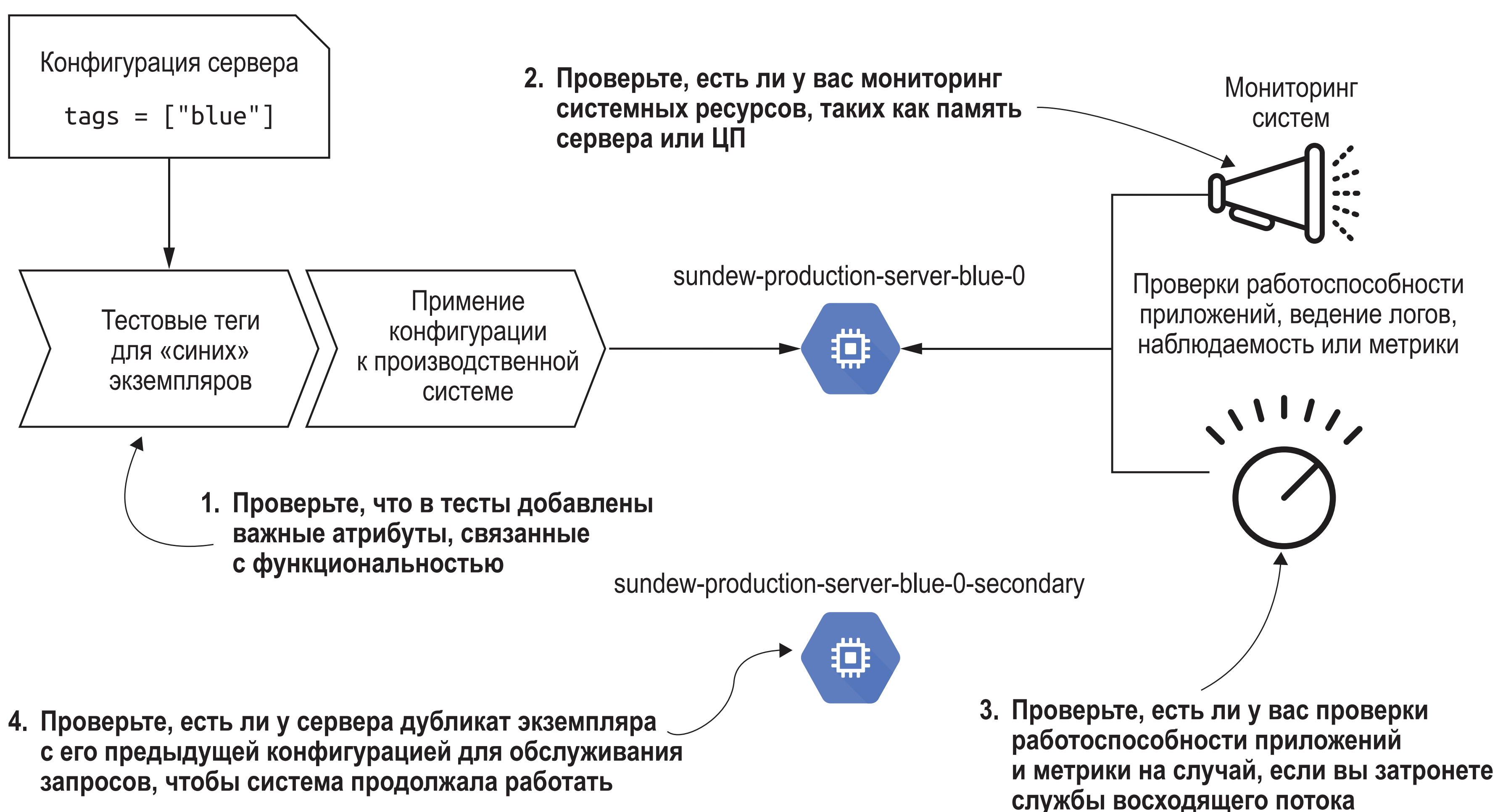


Рис. 9.2 Перед обновлением сети необходимо проверить охват тестов, мониторинг систем и приложений, а также наличие избыточности

Зачем создавать дубликаты серверов? Полезно иметь дополнительные ресурсы, дублирующие предыдущую конфигурацию, кото-

рые вы используете только в случае сбоя основных ресурсов. Эта избыточность поддерживает работоспособность вашей системы.

ОПРЕДЕЛЕНИЕ *Избыточность* (redundancy) – это дублирование ресурсов для повышения производительности системы. В случае сбоя обновленных компонентов система может использовать рабочие ресурсы с предыдущей конфигурацией.

Прежде чем вносить изменения, попробуйте ответить на вопросы из следующего контрольного списка:

- можете ли вы просмотреть каждое изменение и протестировать его в изолированной среде?
- настроены ли в системе *мониторинг и оповещения* для выявления каких-либо аномалий?
- отслеживает ли приложение ошибки с помощью *проверок работоспособности, ведения логов, наблюдения или контроля показателей*?
- имеют ли приложения и их системы какую-либо избыточность?

Пункты в этом списке акцентируют наше внимание на наблюдаемости и осведомленности. Без систем мониторинга или тестов, помогающих выявлять проблемы, у вас могут возникнуть проблемы с выявлением или устранением неработающих изменений. Однажды я внесла изменение, которое вывело из строя приложение, и узнала об этом только через две недели после выпуска. На осознание проблемы ушло так много времени, потому что у нас не было оповещений в приложении!

Контрольный список вопросов, на которые нужно ответить перед изменением системы, закладывает основу для устранения любых проблем и создания планов резервного копирования в случае неудачного обновления. Вы даже можете использовать методы из глав 6 и 8, чтобы включить этот список в конвейеры доставки в качестве входного контроля качества.

9.1.2 *Повышение надежности*

Изучив контрольный список перед изменением, вы приходите к выводу, что сначала нужно улучшить резервное копирование в вашей системе. Чтобы не вывести из строя всю систему выращивания росянки в ходе работ по рефакторингу, вам нужна дополнительная избыточность. Затем, когда вы приступите к внесению изменений в модули команды Cape Sundew, вам не придется беспокоиться о нарушении работы системы.

К сожалению, система выращивания росянки существует только в us-central1. Росянки останутся без полива, если этот регион будет недоступен! Вы решили построить простаивающую систему поли-

ва росянки в другом регионе (`us-west1`), чтобы можно было в случае сбоя переключить на нее приложение для полива. Вы применяете IaC для воспроизведения активной области `us-central1` в пассивной (бездействующей) области `us-west1`.

Теперь вы можете использовать среду в резервном регионе в качестве резервной копии. На рис. 9.3 показано, что вы обновляете конфигурацию команды `Cape Sundew` для использования серверного модуля и отправляете изменения в основную среду. Если это обновление не работает, вы временно отправляете весь трафик в резервную среду, а сами устраняете проблему. Если обновление сработало нормально, вы запускаете тесты и обновляете резервную среду до нового состояния модуля.

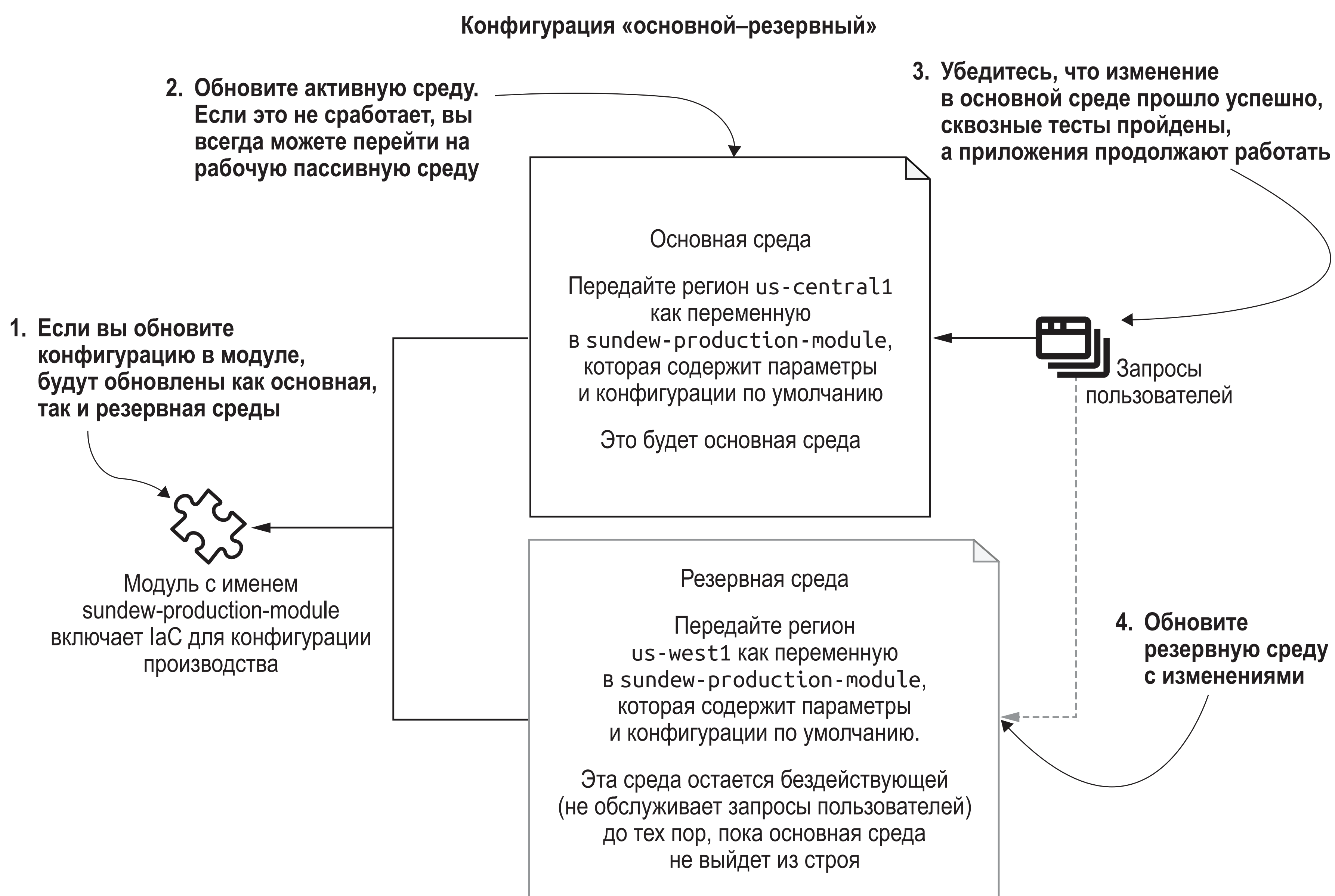


Рис. 9.3 Применение подхода IaC для реализации конфигурации «основной–резервный» системы полива росянки, чтобы повысить ее надежность во время изменений

Система полива росянки теперь использует конфигурацию, в которой одна среда простаивает и служит резервной.

ОПРЕДЕЛЕНИЕ В конфигурации «основной–резервный» одна система является активной средой для выполнения запросов пользователей, а другая – средой резервного копирования.

Если среда в `us-central1` перестанет работать, вы всегда можете отправить трафик в резервную среду `us-west1`. Переключение из вы-

шедшей из строя основной среды в рабочую резервную среду обычно выполняется аварийным переключателем.

ОПРЕДЕЛЕНИЕ *Аварийное переключение* – это практика перехода на пассивные (т. е. резервные) ресурсы в случае сбоя основных ресурсов.

Зачем в этом примере нужна комбинированная конфигурация? Создание резервной среды во втором регионе повышает общую надежность системы. Надежность говорит о том, как долго система работает правильно в течение определенного периода времени.

ОПРЕДЕЛЕНИЕ *Надежность* (reliability) является мерой того, как долго система работает без сбоя в течение определенного периода времени.

Очевидно, вам нужно, чтобы ваша система оставалась надежной при внесении изменений в IaC. Повышение надежности сводит к минимуму сбои в работе критически важных бизнес-приложений и в конечном итоге конечных пользователей. Вы уменьшаете потенциальный ущерб в активной среде с возможностью перенаправления трафика в резервную среду.

Давайте создадим комбинированную конфигурацию в коде. В своем терминале скопируйте файл с именем blue.py, содержащий ресурсы инфраструктуры для выращивания росянки, в новый файл с именем passive.py:

```
$ cp blue.py passive.py
```

В файле passive.py в листинге 9.1 мы обновляем несколько переменных, чтобы создать резервную среду управления поливом росянки, включая регион и имя.

Листинг 9.1 Добавление резервной среды us-west1 для команды выращивания росянки

```
TEAM = 'sundew'
ENVIRONMENT = 'production'
VERSION = 'passive'
REGION = 'us-west1'
IP_RANGE = '10.0.1.0/24'

zone = f'{REGION}-a'
network_name = f'{TEAM}-{ENVIRONMENT}-network-{VERSION}'
server_name = f'{TEAM}-{ENVIRONMENT}-server-{VERSION}'

cluster_name = f'{TEAM}-{ENVIRONMENT}-cluster-{VERSION}'
cluster_nodes = f'{TEAM}-{ENVIRONMENT}-cluster-nodes-{VERSION}'
cluster_service_account = f'{TEAM}-{ENVIRONMENT}-sa-{VERSION}'
```



```
labels = {
    'team': TEAM,
    'environment': ENVIRONMENT,
    'automated': True
}
```

- ❶ Устанавливаем версию для идентификации резервной среды.
- ❷ Изменяем регион с `us-central1` на `us-west1` для резервной среды.
- ❸ Задаем другой диапазон IP-адресов для резервной среды, чтобы избежать отправки запросов.
- ❹ Остальные переменные и функции ссылаются на константы резервной среды, включая регион.
- ❺ Определяем метки для ресурсов, чтобы вы могли отличить промышленную среду.

Эквиваленты AWS и Azure

Метки GCP аналогичны тегам AWS и Azure. Вы можете взять объект, определенный в переменной `labels`, и передать его в теги ресурсов AWS и Azure.

Теперь у вас есть резервная среда на случай, если изменение вашего модуля пойдет не так. Представьте, что вы вносите изменения и нарушаете работу основной среды в `us-central1`. Вы можете перенастроить конфигурацию производственного глобального балансировщика нагрузки на устранение отказа и перенаправить весь трафик в резервную среду. В следующем листинге мы изменяем настройку глобального балансировщика нагрузки, чтобы отправлять 100 % трафика в резервную среду.

Листинг 9.2 Переключение на резервную среду поддержки выращивания росянки в `us-west1`

```
import blue
import passive

services_list = [
    {
        'version': 'blue',
        'zone': blue.zone,
        'name': f'{shared_name}-blue',
        'weight': ,0
    }, {
        'version': 'passive',
        'zone': passive.zone,
        'name': f'{shared_name}-passive',
        'weight': 100
    }
]

def _generate_backend_services(services):
```



```

backend_services_list = []
for service in services:
    version = service['version']
    weight = service['weight']
    backend_services_list.append({
        'backend_service': (
            '${google_compute_backend_service.'
            f'{version}.id}}'
        ),
        'weight': weight,
    })
return backend_services_list

def load_balancer(name, default_version, services):
    return [{
        'google_compute_url_map': {
            TEAM: [{
                'name': name,
                'path_matcher': [{
                    'name': 'allpaths',
                    'path_rule': [{
                        'paths': [
                            '/*'
                        ],
                        'route_action': {
                            'weighted_backend_services':
                                _generate_backend_services(
                                    services)
                        }
                    ]
                }]
            }]
        }
    }]
}

```

- 1 Импортируем IaC как для синей (основной среды), так и для резервной среды.
- 2 Определяем список версий для каждой среды, подключаемой к балансировщику нагрузки, – синей и резервной версий.
- 3 Настраиваем балансировщик нагрузки для отправки 0 % трафика в синюю версию.
- 4 Настраиваем балансировщик нагрузки для отправки 100 % трафика в резервную версию.
- 5 Добавляем две версии в балансировщик нагрузки с их весами в качестве указателей на правила балансировки нагрузки.
- 6 Определяем серверные службы для синей и резервной сред для направления трафика в каждую из них.
- 7 Создаем карту URL-адресов вычислений Google (правило балансировки нагрузки) с использованием ресурса Terraform на основе пути, синего (рабочего) и резервного серверов, а также веса (доли трафика).
- 8 Настраиваем правило пути, которое направляет все пути к основным или резервным серверам.
- 9 Добавляем две версии в балансировщик нагрузки с их весами в качестве указателя на правила балансировки нагрузки.

Эквиваленты AWS и Azure

Карта сетевых адресов Google Cloud аналогична AWS Application Load Balancer (ALB) или Azure Traffic Manager and Application Gateway. Чтобы перенести листинг 9.2 в AWS, вам необходимо обновить ресурс для создания правила AWS ALB и прослушивателя. Затем добавьте атрибуты маршрутизации пути и веса в правило прослушивателя ALB.

Для Azure вам потребуется связать профиль и конечную точку диспетчера трафика Azure со шлюзом приложений Azure. Обновите диспетчер трафика Azure, указав веса, и направьте их в правильный пул внутренних адресов, подключенный к шлюзу приложений Azure.

После переключения системы на резервную (пассивную) среду команда Cape Sundew сообщает о восстановлении работы приложения для полива росянки. У вас есть возможность спокойно отладить проблемы с вашим модулем в синей (активной) среде. Комбинированная конфигурация защитит от сбоев в отдельных регионах в будущем.

Члены команды Cape Sundew говорят вам, что в конечном итоге они хотят направить трафик в оба региона, чтобы обе среды обрабатывали запросы. На рис. 9.4 показана конфигурация их мечты. При следующем изменении модуля они развернут обновления только

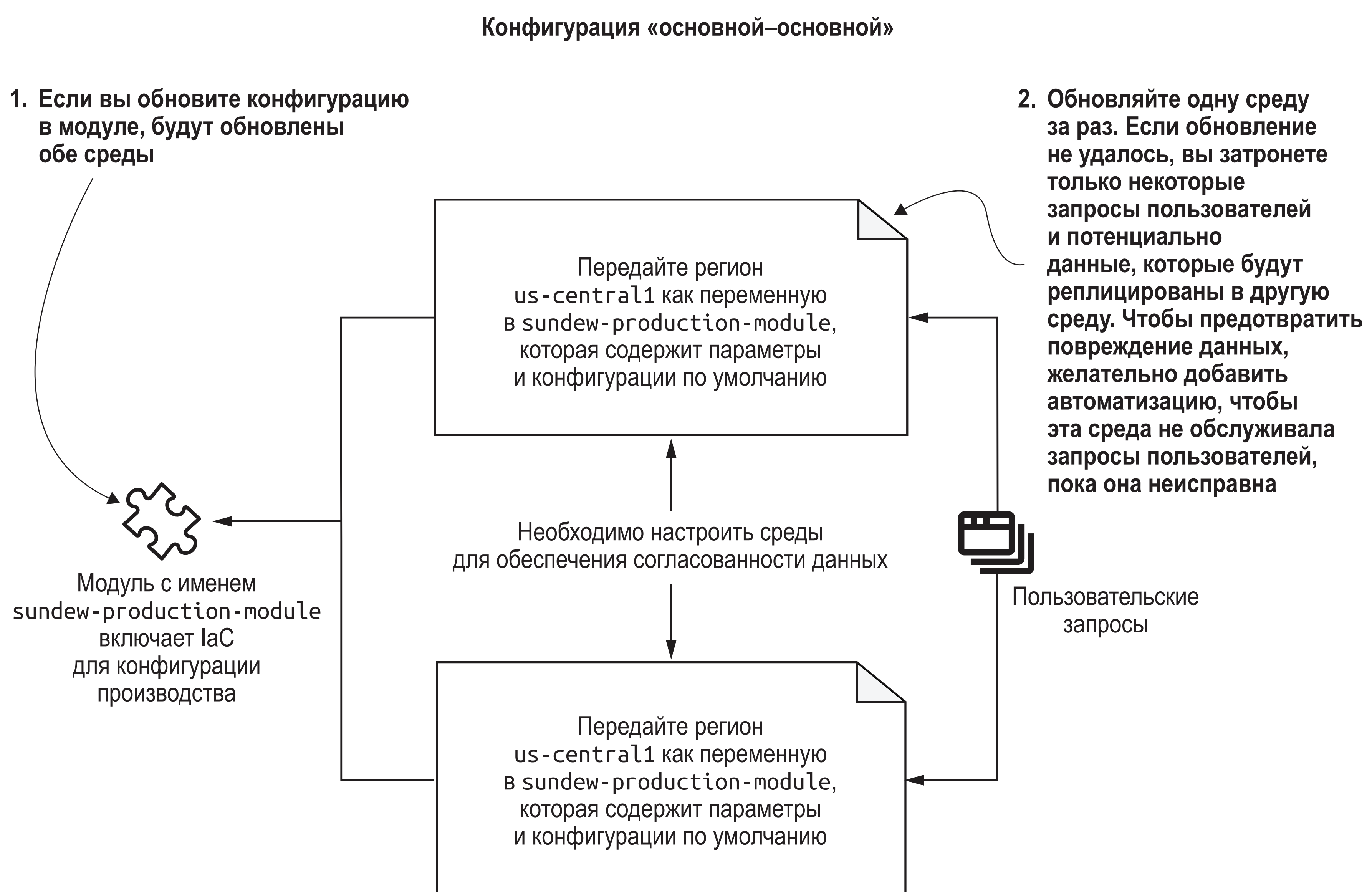


Рис. 9.4 В будущем команда Cape Sundew проведет дальнейший рефакторинг конфигурации системы для поддержки схемы «основной–основной» и направит запросы в оба региона

в одном регионе. Если регион выйдет из строя, большинство запросов все равно будут обрабатываться системой. Вы реже поливаете росянку, но у вас есть возможность устранить неполадку.

Зачем стремиться к использованию двух основных сред? Многие распределенные системы работают именно в такой конфигурации, когда оба сегмента обрабатывают и принимают запросы, а также реплицируют данные между собой. В архитектурах публичного облака для повышения надежности системы рекомендуется использовать конфигурацию «основной–основной», состоящую из нескольких регионов.

ОПРЕДЕЛЕНИЕ В конфигурации «основной–основной» несколько систем являются основной средой для выполнения запросов пользователей и репликации данных между ними.

Ваши изменения в IaC зависят от варианта конфигурации – «основной–резервный» или «основной–основной». Вторая конфигурация команды Cape Sundew должна реорганизовать IaC в более модульные компоненты и поддерживать репликацию данных между средами. Поскольку известно, что команда Cape Sundew проведет рефакторинг своих приложений для поддержки конфигурации «основной–основной», вам потребуется реализовать какую-то форму глобальной балансировки нагрузки в вашей конфигурации IaC и подключить ее к каждому региону.

Технология IaC соответствует принципу воспроизводимости, о котором мы говорили в главе 1. Благодаря этому принципу мы можем создать новую среду в новом регионе с несколькими обновлениями атрибутов. Вам не придется кропотливо перестраивать ресурс за ресурсом, как мы это делали в главе 2.

Однако вы можете столкнуться с необходимостью копировать и вставлять множество одинаковых конфигураций. Попробуйте передавать регионы в качестве входных данных и разбить IaC на модули, чтобы уменьшить количество копирований-вставок. Отделите конфигурацию общих ресурсов, например определения глобальной балансировки нагрузки, от конфигураций среды.

Мультирегиональные (многозонные) среды всегда нуждаются в дополнительных затратах денег и времени, но повышают надежность системы. IaC ускоряет создание копий новых сред в других регионах и обеспечивает согласованность конфигураций. Несогласованность между регионами может привести к серьезным сбоям в системе и увеличению затрат на обслуживание! В главе 12 обсуждается управление затратами и его аспекты.

9.2 *Blue-green развертывание*

Теперь, когда у вас есть резервная среда, к которой можно обратиться, если вы случайно повредите активную, вы можете начать обнов-

ление системы выращивания росянки для использования глобальной балансировки и для высокой доступности серверов. Система выращивания росянки использует активно-пассивную конфигурацию, что означает дублирование совершенно новой среды в новом регионе только для внесения изменений.

Вы понимаете, что некоторые изменения не требуют дублирования всей среды в нескольких регионах. Зачем нужна целая пассивная среда только для обновления сервера? Неужели нельзя просто обновить один сервер? В конце концов, мы хотим не только свести к минимуму потенциальный ущерб от сбоя, но и оптимизировать эффективность использования ресурсов. Вместо использования активно-пассивной конфигурации вы можете применить этот подход к меньшему количеству ресурсов в гораздо меньшем масштабе.

На рис. 9.5 показано, что вы воспроизводите новый *сегмент* с глобальным доступом вместо всей среды. Вы помечаете новый сегмент тегом *ggreen* (зеленый) и развертываете в нем набор из трех серверов и одного кластера. После тестирования новых ресурсов вы используете глобальный балансировщик нагрузки для отправки небольшого процента трафика на новые ресурсы. Запросы выполняются успешно, значит, обновление глобальной маршрутизации сработало.

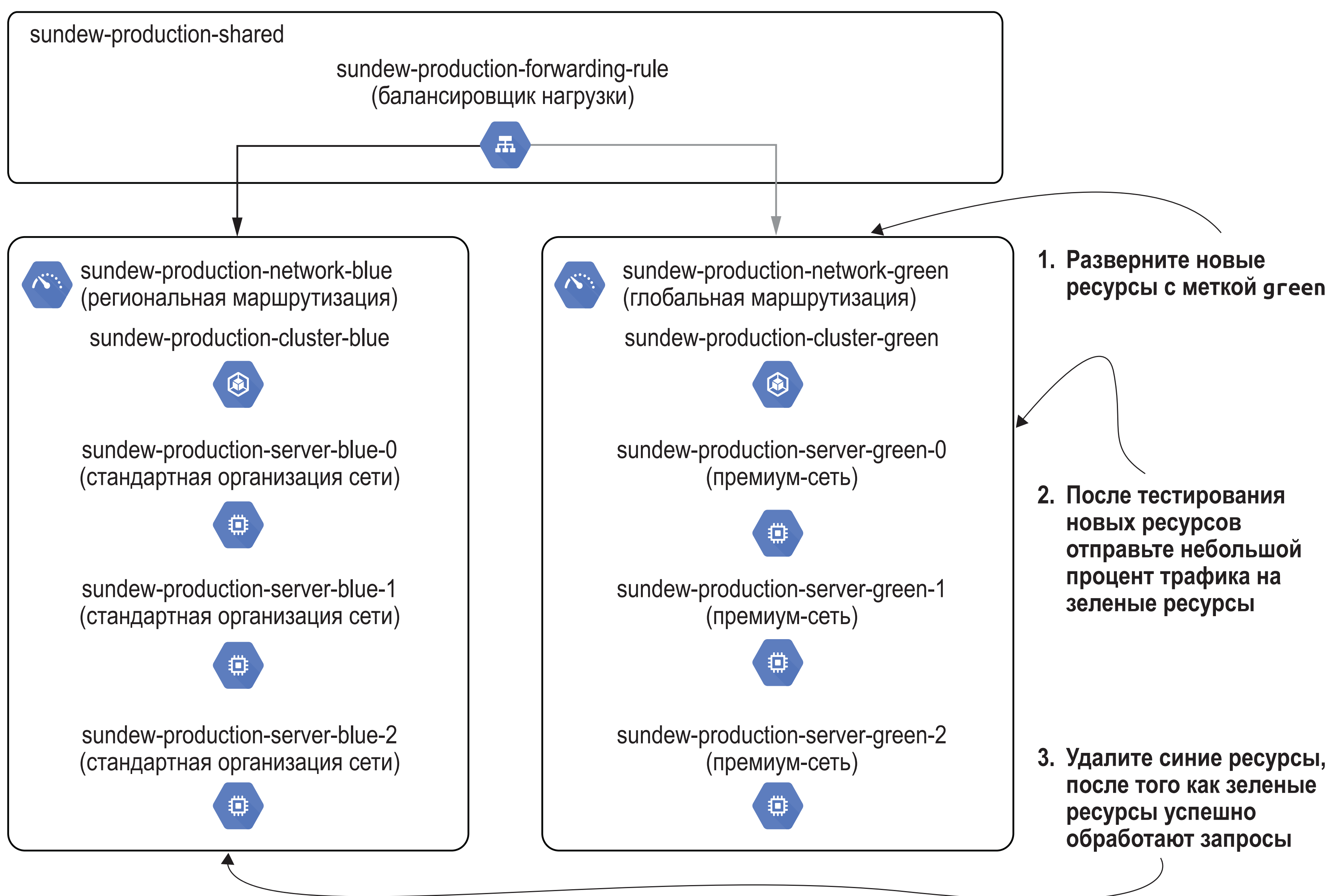


Рис. 9.5 Используйте blue-green развертывание, чтобы создать зеленую среду для глобальной сети, отправить часть трафика на новые ресурсы и удалить старые ресурсы

Схема создания нового набора ресурсов и постепенного перехода к ним основана на принципах компонуемости и развиваемости системы. Вы добавляете в свою среду новый зеленый набор ресурсов green и позволяете им развиваться независимо от старых ресурсов. Если вы хотите внести изменения в инфраструктуру, повторите этот рабочий процесс, чтобы уменьшить потенциальный ущерб от изменений и протестировать новые ресурсы перед передачей им части трафика.

Этот паттерн, называемый *blue-green развертыванием*, создает новое подмножество ресурсов инфраструктуры с тегом green, которое реализует необходимые изменения. Затем вы направляете несколько запросов к ресурсам промежуточной зеленой инфраструктуры и проверяете, все ли работает. Постепенно вы перенаправляете все запросы к ресурсам зеленой инфраструктуры и удаляете старые производственные ресурсы, помеченные тегом blue.

ОПРЕДЕЛЕНИЕ *Blue-green развертывание* (blue-green deployment) – это паттерн, согласно которому вы создаете новое подмножество ресурсов инфраструктуры с изменениями, которые вы хотите внести. Вы постепенно перемещаете трафик со старого набора ресурсов (синий) на новый набор ресурсов (зеленый) и в конечном итоге удаляете старые ресурсы.

Blue-green развертывание позволяет изолировать и тестировать изменения в (временно) новой промежуточной среде перед отправкой к ней запросов. После проверки зеленой среды вы можете сделать новую среду производственной и удалить старую. Вы временно платите за две среды в течение нескольких недель, но минимизируете общие затраты на обслуживание постоянных сред.

ПРИМЕЧАНИЕ Blue-green развертывание имеет несколько разных меток, иногда с тонкими нюансами в зависимости от контекста. Не имеет значения, каким цветом или именем вы помечаете среды, главное, чтобы вы могли определить, какая среда является производственной или промежуточной. Я также использую метки production/staging (производственная/промежуточная) и нумерацию версий (v1/v2) для различения старых и новых ресурсов во время blue-green развертывания.

Вам следует использовать blue-green паттерн развертывания для рефакторинга или обновления вашего IaC за пределами нескольких минимальных конфигураций. Такое развертывание следует принципу неизменности при создании новых ресурсов, постепенной передаче им трафика или функциональности и удалении старых ре-

сурсов. Большинство паттернов рефакторинга (глава 10) и изменения IaC основаны на принципе воспроизводимости.

Создание образа и управление конфигурацией

Аналогичным образом с помощью blue-green развертывания можно снизить риск сбоя, вызванного неудачным образом операционной системы или конфигурации сервера. Изолируйте обновления образа машины или управления конфигурацией на новом сервере (зеленый), направьте на него трафик и проверьте его функциональность перед удалением старого сервера (синий).

У вас уже есть глобальный балансировщик нагрузки для существующей синей сети, который вы сможете использовать позже для подключения новой зеленой сети. В следующих разделах мы реализуем каждый шаг blue-green развертывания системы выращивания росянки.

9.2.1 Развертывание зеленой инфраструктуры

Чтобы начать blue-green развертывание системы выращивания росянки и серверов с высокой доступностью, вы копируете конфигурацию существующего синего сегмента. Вы создаете файл с именем `green.py` и вставляете синюю конфигурацию сегмента. В листинге 9.3 показаны изменения в определении сегмента, чтобы оно задавало режим глобальной маршрутизации.

Листинг 9.3 Создание зеленого сегмента

```
TEAM = 'sundew'
ENVIRONMENT = 'production'
VERSION = 'green' ❶
REGION = 'us-central1'
IP_RANGE = '10.0.0.0/24' ❷

zone = f'{REGION}-a'
network_name = f'{TEAM}-{ENVIRONMENT}-network-{VERSION}' ❸

labels = {
    'team': TEAM,
    'environment': ENVIRONMENT,
    'automated': True
}

def build(): ❸
    return network() ❸

def network(name=network_name, ❸
            region=REGION,
```



```

        ip_range=IP_RANGE):
return [
    {
        'google_compute_network': {
            VERSION: [{
                'name': name,
                'auto_create_subnetworks': False,
                'routing_mode': 'GLOBAL'
            }]
        },
        {
            'google_compute_subnetwork': {
                VERSION: [{
                    'name': f'{name}-subnet',
                    'region': region,
                    'network': f'${google_compute_network.{VERSION}.name}',
                    'ip_cidr_range': ip_range
                }]
            }
        }
    ]

```

- ❶ Задаем имя новой версии сети 'green'.
- ❷ Сохраняем диапазон IP-адресов для зеленой сети таким же, как и для синей. GCP позволяет двум сетям иметь один и тот же блок CIDR, если вы не настроили пиринг.
- ❸ Применяем модуль для создания конфигурации JSON сети и подсети для зеленой сети.
- ❹ Создаем сеть Google с помощью ресурса Terraform на основе имени и режима глобальной маршрутизации.
- ❺ Меняем режим маршрутизации зеленой сети на глобальный, чтобы предоставлять маршруты глобально.
- ❻ Создаем подсеть Google с использованием ресурса Terraform на основе имени, региона, сети и диапазона IP-адресов.

Эквиваленты AWS и Azure

Если вы переносите код листинга 9.3 в AWS или Azure, режим глобальной маршрутизации не применяется. Вы по-прежнему можете обновить код для AWS или Azure, изменив сеть и подсеть Google на VPC или виртуальную сеть, а также изменив подсети и таблицы маршрутизации.

По возможности желательно сохранить одну и ту же конфигурацию для синих и зеленых ресурсов. Они должны отличаться только изменениями, которые вы хотите внести в зеленые ресурсы. Тем не менее у вас могут быть и другие различия!

Например, если бы у меня была какая-то конкретная конфигурация пиринга для моих сетей, я не смогла бы использовать диапазон IP-адресов синей сети для зеленой сети. Вместо этого мне понадобится другой диапазон IP-адресов, например 10.0.1.0/24, поэтому придется обновить все зависимости для связи с новым диапазоном.

Blue-green развертывание способствует неизменности, созданию новых ресурсов и изоляции изменений от старых ресурсов. Однако развертывание новой версии низкоуровневого ресурса, такого как сеть, не означает, что вы можете немедленно отправлять на него живой трафик. Вы всегда начинаете с изменения и тестирования ресурса инфраструктуры, который хотите обновить. Затем вы должны изменить и протестировать другие ресурсы, которые от него зависят.

9.2.2 Развертывание высокоуровневых зависимостей в зеленой инфраструктуре

При использовании blue-green развертывания всегда необходимо развертывать новый ресурс инфраструктуры с изменениями и новый набор ресурсов высокого уровня, которые от него зависят. Вы завершили обновление сети, но не можете использовать ее, если в ней нет серверов и приложений. Новой сети нужна инфраструктура высокого уровня, которая от нее зависит.

Вы связываетесь с командой Cap Sundew и просите развернуть новые кластеры и серверы в зеленой сети, как показано на рис. 9.6. Серверы должны использовать сеть с высокой доступностью с публичным доступом. Команда Cap Sundew также развертывает свои приложения на кластере и серверах.

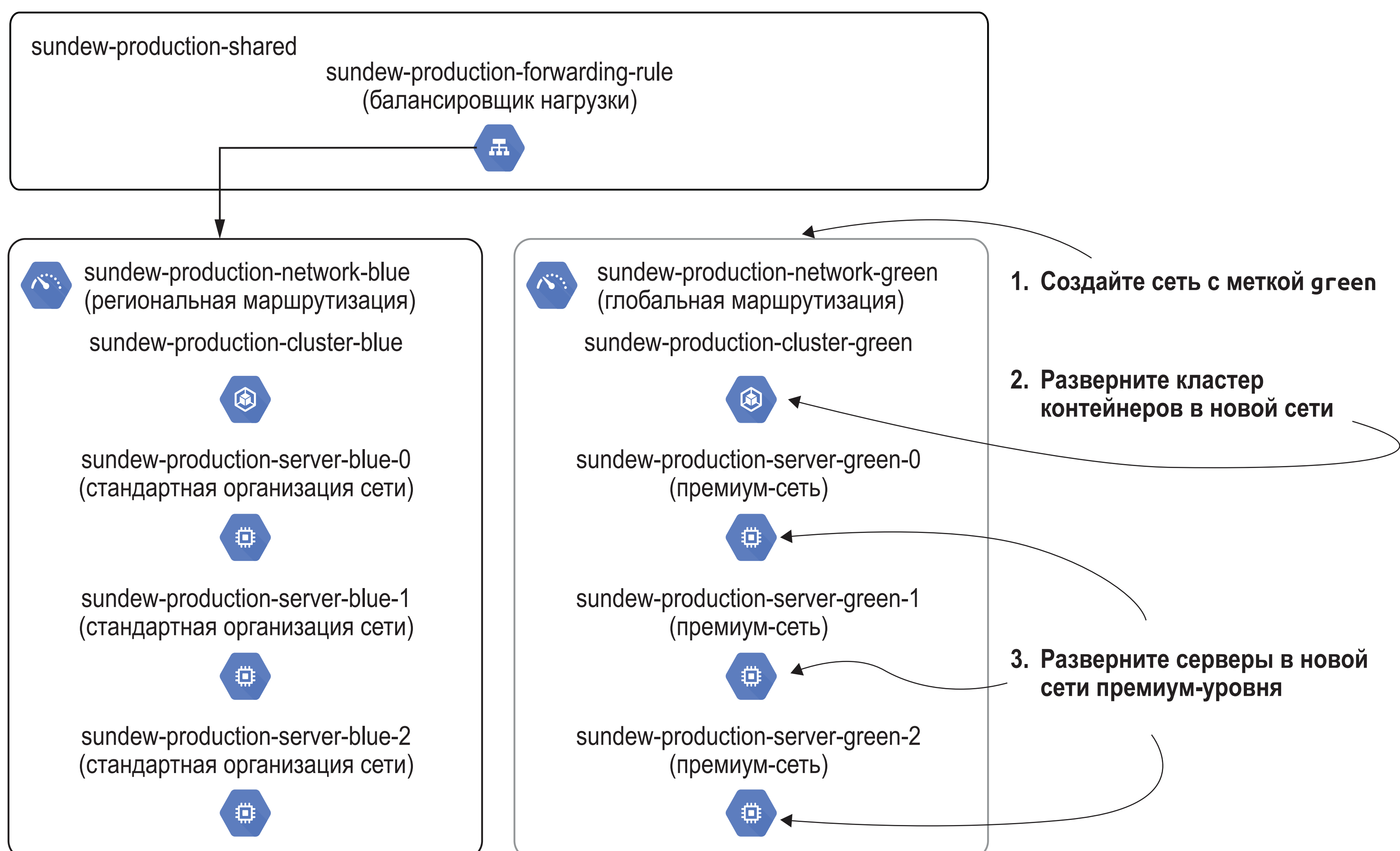


Рис. 9.6 После создания ресурса инфраструктуры низкого уровня, такого как сеть, вам также необходимо создать новые ресурсы высокого уровня, такие как сервер и кластер контейнеров, которые зависят от него

В этом примере изменение ресурса инфраструктуры низкого уровня, такого как сеть, влияет на ресурсы высокого уровня. Серверы должны работать с высокодоступной сетью. Обновление исходной синей сети с региональной на глобальную маршрутизацию, скорее всего, повлияет на серверы и кластер. При blue-green развертывании вы развиваете сетевые атрибуты серверов, не затрагивая производственную среду.

Рассмотрим образец IaC, который члены команды Cap Sundew добавили для развертывания кластера в зеленой сети (листинг 9.4). Они скопировали конфигурацию кластера из ресурса blue и обновили его атрибуты для работы в сети green.

Листинг 9.4 Добавление нового кластера в зеленую сеть

```

VERSION = 'green' 1

cluster_name = f'{TEAM}-{ENVIRONMENT}-cluster-{VERSION}' 1
cluster_nodes = f'{TEAM}-{ENVIRONMENT}-cluster-nodes-{VERSION}' 1
cluster_service_account = f'{TEAM}-{ENVIRONMENT}-sa-{VERSION}' 1

def build(): 2
    return network() + \
        cluster() 3

def cluster(name=cluster_name, 4
            node_name=cluster_nodes, 4
            service_account=cluster_service_account, 4
            region=REGION): 4
    return [
        {
            'google_container_cluster': { 5
                VERSION: [
                    {
                        'initial_node_count': 1,
                        'location': region,
                        'name': name, 5
                        'remove_default_node_pool': True,
                        'network':
                            f'${{google_compute_network.{VERSION}.name}}', 6
                        'subnetwork': \ 6
                            f'${{google_compute_subnetwork.{VERSION}.name}}' 6
                    }
                ]
            }
        ]
    ]

```

- ❶ Помечаем новую версию кластера как 'green'.
- ❷ Применяем модуль для создания конфигурации JSON сети, подсети и кластера для зеленой сети.
- ❸ Создаем кластер в зеленой сети и подсети.

- ④ Передаем в кластер необходимые атрибуты, включая имя, имена узлов, учетные записи служб для автоматизации и регион.
- ⑤ Создаем кластер контейнеров Google с использованием ресурса Terraform с одним узлом и в зеленой сети.
- ⑥ Создаем кластер в зеленой сети и подсети.

Эквиваленты AWS и Azure

Вы можете адаптировать код, изменив кластер контейнера Google на кластер Amazon EKS или кластер Azure Kubernetes Service (AKS). Вам понадобятся Amazon VPC и виртуальная сеть Azure для пулов узлов Kubernetes (также называемых *группами*).

Кластер не требует каких-либо изменений для адаптации к конфигурации публичной сети. Однако серверам необходима высоко-доступная сеть. Скопируем конфигурацию сервера из синего цвета и изменим ее, чтобы использовать сетевые атрибуты высокой доступности в green.py (листинг 9.5).

Листинг 9.5 Добавление сети высокой доступности к серверам в зеленой сети

```

VERSION = 'green' ①

server_name = f'{TEAM}-{ENVIRONMENT}-server-{VERSION}' ②

def build(): ③
    return network() + \
        cluster() + \
        server0() + \ ④
        server1() + \ ④
        server2() ④

def server0(name=f'{server_name}-0', ⑤
            zone=zone): ⑤
    return [
        {
            'google_compute_instance': { ⑥
                f'{VERSION}_0': [{
                    'allow_stopping_for_update': True,
                    'boot_disk': [{
                        'initialize_params': [{
                            'image': 'ubuntu-1804-lts'
                        }]
                    }],
                    'machine_type': 'f1-micro',
                    'name': name,
                    'zone': zone,
                    'network_interface': [{
                        'subnetwork': \
                            f'${{google_compute_subnetwork.{VERSION}.name}}',

```



```

        'access_config': {
            'network_tier': 'PREMIUM'
        }
    }
}
]

```

- ❶ Помечаем новую версию сети как 'green'.
- ❷ Создаем паттерн имени сервера, который включает команду, среду и версию (blue или green).
- ❸ Применяем модуль для создания конфигурации JSON сети, подсети, кластера и сервера для зеленой сети.
- ❹ Создаем кластер из трех серверов в зеленой сети.
- ❺ Копируем и вставляем каждую конфигурацию сервера. В этом фрагменте кода представлен первый сервер `server0`. Конфигурации других серверов опущены для ясности.
- ❻ Создаем небольшой вычислительный экземпляр (сервер) Google с использованием ресурса Terraform в зеленой сети.
- ❼ Задаем уровень сети премиум. Это обеспечивает совместимость с базовой подсетью, которая использует глобальную маршрутизацию.

Эквиваленты AWS и Azure

Если вы переносите код листинга 9.5 в AWS или Azure, сетевой уровень *не применяется*. Вы по-прежнему можете обновить код, изменив вычислительный экземпляр Google на экземпляр Amazon EC2 или виртуальную машину Azure Linux с образом Ubuntu 18.04. Сначала вам понадобится Amazon VPC и виртуальная сеть Azure.

Обновление сетевого уровня до высокодоступного не должно повлиять на функциональность приложений, хотя вы не можете знать этого наверняка! Зеленая среда позволяет выявлять и смягчать любые проблемы, прежде чем они повлияют на рост росянки. После того как команда Cap Sundew внесет изменения, она выпускает обновление и проверяет результаты тестирования в конвейере поставки.

Команда выполняет модульное, интеграционное и сквозное тестирование, чтобы убедиться, что сможет запускать приложения в новом кластере контейнеров и отправлять запросы на новые зеленые серверы. К счастью, тесты пройдены, и все готово для отправки живого трафика на зеленые ресурсы.

9.2.3 Использование канареечного развертывания в зеленой инфраструктуре

Вы могли бы сразу же отправить весь трафик в зеленую сеть, на серверы и в кластер. Но вы не хотите полностью обрушить бизнес

по выращиванию росянки, если что-то пойдет не так! В идеале вам нужно иметь возможность переключить весь трафик на синюю среду, если развернутое в зеленой среде обновление вызовет сбой. На рис. 9.7 вы настраиваете глобальный балансировщик нагрузки так, чтобы он отправлял 90 % трафика в синюю сеть и 10 % – в зеленую.

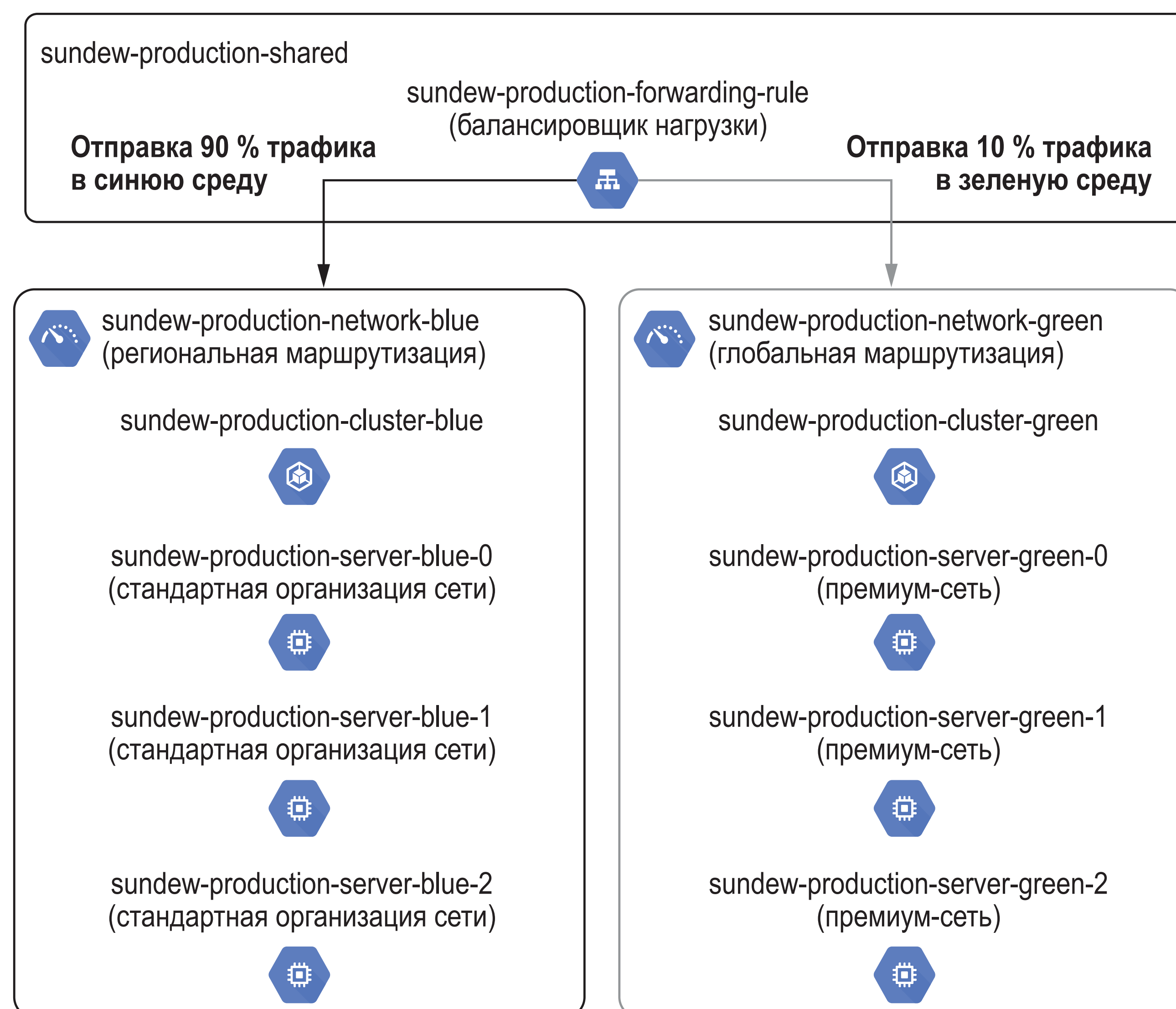


Рис. 9.7 Настройка глобального балансировщика нагрузки для запуска канареечного теста и отправки небольшого процента трафика на зеленые ресурсы

Отправка небольшого пробного трафика в обновленную систему называется канареечным развертыванием. Если эта пробная нагрузка приводит к ошибкам в запросах, вам необходимо отладить и исправить обновления.

ОПРЕДЕЛЕНИЕ *Канареечное развертывание* (canary deployment) – это паттерн, в соответствии с которым мы отправляем небольшой процент трафика на обновленные ресурсы в системе. Если запросы выполняются успешно, мы постепенно увеличиваем долю трафика.

Зачем сначала отправлять небольшой объем трафика? Было бы неразумно подвергать риску сбой все запросы. Отправка нескольких запросов к обновленным ресурсам помогает выявить критические проблемы, прежде чем они повлияют на всю вашу систему.

Канарейка в угольной шахте

Канарейка в программном обеспечении или инфраструктуре служит первым индикатором того, будет ли работать ваша новая система, функция или приложение. Этот термин происходит от выражения «канарейка в угольной шахте». Шахтеры приносили с собой в шахты птиц в клетках. Если в воздухе шахты появлялась примесь опасного газа, то канарейки сразу умирали, сигнализируя своей гибелью об опасности.

В разработке программного обеспечения вы часто встретите упоминания о канареечном тестировании, которое измеряет удобство использования новой версии приложения или функции. Я настоятельно рекомендую применять канареечное развертывание – метод отправки небольшого процента трафика на новые ресурсы каждый раз, когда вы вносите существенные изменения в инфраструктуру.

Стоит отметить, что для канареечного развертывания *не обязательно* использовать балансировщики нагрузки. Вы можете применить любой метод отправки нескольких запросов к обновляемым ресурсам инфраструктуры. Например, вы можете добавить один обновленный экземпляр приложения в существующий пул из трех экземпляров приложения. Схема балансировки нагрузки с циклическим перебором отправляет около 25 % ваших запросов в новый обновленный экземпляр и 75 % – в старые существующие экземпляры приложения.

Что касается команды Cap Sundew, вы отделяете глобальную конфигурацию балансировщика нагрузки от зеленой и синей сред в вашей конфигурации. Это улучшает возможность развития балансировщика нагрузки. Вы добавляете зеленые серверы в качестве отдельной серверной службы в балансировщик нагрузки и управляете распределением запросов между зеленой и синей средами.

Балансировщик нагрузки определяется в файле с именем `shared.ru` в листинге 9.6. Добавим зеленую версию сети (серверов и кластера тоже!) в перечень версий сред с весом 10.

Листинг 9.6 Добавление зеленой версии в список сервисов балансировки нагрузки

```
import blue 1
import green 1

shared_name = f'{TEAM}-{ENVIRONMENT}-shared' 2

services_list = [ 3
    {
        'version': 'blue', 4
        'zone': blue.zone, 4
        'name': f'{shared_name}-blue', 4
    }
```



```

        'weight': 90
    },
    {
        'version': 'green',
        'zone': green.zone,
        'name': f'{shared_name}-green',
        'weight': 10
    }
]

def _generate_backend_services(services):
    backend_services_list = []
    for service in services:
        version = service['version']
        weight = service['weight']
        backend_services_list.append({
            'backend_service': (
                '${google_compute_backend_service.'
                f'{version}.id}}'
            ),
            'weight': weight,
        })
    return backend_services_list

```

- ❶ Импортируем IaC как для синей, так и для зеленой среды.
- ❷ Формируем имя для общего глобального балансировщика нагрузки на основе имени команды и среды.
- ❸ Определяем список версий для каждой среды для подключения к балансировщику нагрузки, синей и зеленой средам.
- ❹ Добавляем синюю сеть, серверы и кластер в список балансировщиков нагрузки. Получаем зону доступности синей среды из ее IaC.
- ❺ Устанавливаем вес трафика к синим экземплярам сервера равным 90, что соответствует 90 % запросов.
- ❻ Добавляем зеленую сеть, серверы и кластер в список балансировщиков нагрузки. Получаем зону доступности зеленой среды из ее IaC.
- ❼ Устанавливаем вес трафика к экземплярам зеленого сервера равным 10, что соответствует 10 % запросов.
- ❽ Добавляем функцию для создания списка серверных служб для балансировщика нагрузки.
- ❾ Для каждой среды определяем серверную службу балансировки нагрузки Google с версией и весом.

Эквиваленты AWS и Azure

Серверные службы в листинге 9.6 аналогичны целевой группе AWS для AWS ALB. Однако Azure требует дополнительных ресурсов. Вам потребуется создать профиль диспетчера трафика Azure и конечную точку для пула внутренних адресов, подключенного к шлюзу приложений Azure.

Балансировщик нагрузки в файле `shared.py` уже принимает список серверных служб с разным весом. После того как вы развернете

список весов и сервисов, приведенный в листинге 9.7, конфигурация балансировщика нагрузки начнет отправлять 10 % трафика в зеленую сеть.

Листинг 9.7 Обновление балансировщика нагрузки для передачи трафика в зеленую среду

```

default_version = 'blue' ❶

def load_balancer(name, default_version, services): ❷
    return [{
        'google_compute_url_map': { ❸
            'TEAM': [{
                'default_service': ( ❶
                    '${google_compute_backend_service.' ❶
                    f'{default_version}.id}}' ❶
                ),
                'description': f'URL Map for {TEAM}',
                'host_rule': [{
                    'hosts': [
                        f'{TEAM}.{COMPANY}.com'
                    ],
                    'path_matcher': 'allpaths'
                ]},
                'name': name,
                'path_matcher': [{
                    'default_service': ( ❶
                        '${google_compute_backend_service.' ❶
                        f'{default_version}.id}}' ❶
                    ), ❶
                    'name': 'allpaths',
                    'path_rule': [{
                        'paths': [
                            '/'
                        ],
                        'route_action': {
                            'weighted_backend_services': ❷
                                _generate_backend_services( ❷
                                    services) ❷
                        }
                    ]
                ]
            }
        ]
    }]
}

```

- ❶ По умолчанию отправляем весь трафик из балансировщика нагрузки в синюю среду.
- ❷ Применяем модуль для создания конфигурации JSON балансировщика нагрузки, чтобы отправлять 10 % трафика в зеленую среду и 90 % – в синюю.
- ❸ Создаем карту адресов серверов Google (правило балансировки нагрузки) с помощью ресурса Terraform на основе пути, синей и зеленой сред и веса.

- 4 Устанавливаем правила маршрутизации на балансировщике нагрузки, чтобы отправлять 10 % трафика в зеленую и 90 % трафика в синюю.

Эквиваленты AWS и Azure

Карта адресов Google Cloud аналогична AWS ALB или диспетчеру трафика Azure и шлюзу приложений. Чтобы перенести код листинга 9.7 в AWS, вам необходимо обновить ресурс для создания правила AWS ALB и прослушивателя. Затем добавьте атрибуты маршрутизации пути и веса в правило прослушивателя ALB.

Для Azure вам потребуется связать профиль и конечную точку диспетчера трафика Azure со шлюзом приложений Azure. Обновите диспетчер трафика Azure, указав веса, и направьте их в правильный пул внутренних адресов, подключенный к шлюзу приложений Azure.

Далее мы выполняем код Python из листинга 9.8, чтобы создать конфигурацию Terraform в формате JSON для проверки. Конфигурация JSON для балансировщика нагрузки содержит группы экземпляров, которые организуют синие и зеленые серверы, серверные службы, предназначенные для групп синих и зеленых вычислительных экземпляров, а также действия взвешенной маршрутизации.

Листинг 9.8 Конфигурация JSON для балансировщика нагрузки

```
{
  "resource": [
    {
      "google_compute_url_map": {
        "sundew": [
          {
            "default_service": \
              "${google_compute_backend_service.blue.id}",
            "description": "URL Map for sundew",
            "host_rule": [
              {
                "hosts": [
                  "sundew.dc4plants.com"
                ],
                "path_matcher": "allpaths"
              }
            ],
            "name": "sundew-production-shared",
            "path_matcher": [
              {
                "default_service":
                  "${google_compute_backend_service.blue.id}",
                "name": "allpaths",
                "path_rule": [

```



```

    "paths": [
      "/"
    ],
    "route_action": {
      "weighted_backend_services": [
        {
          "backend_service":
            "${google_compute_backend_
            service.blue.id}",
          "weight": 90,
        },
        {
          "backend_service":
            "${google_compute_backend_
            service.green.id}",
          "weight": 10,
        }
      ]
    }
  }
}

```

- 1 Определяем карту адресов вычислительных экземпляров Google (правило балансировки нагрузки) с использованием ресурса Terraform на основе пути, синей и зеленой среды и веса.
- 2 Определяем службу по умолчанию для сопоставления адресов вычислительных экземпляров Google (правило балансировки нагрузки) с синей средой.
- 3 Отправляем все запросы в синюю или зеленую среды в зависимости от веса.
- 4 Отправляем 90 % трафика на синие серверные службы, которые используют синюю сеть.
- 5 Отправляем 10 % трафика на зеленые серверные службы, которые используют зеленую сеть.

Зачем отправлять весь трафик по умолчанию в синюю среду? Вы уверены, что синяя среда успешно обрабатывает запросы. Если ваша зеленая среда выходит из строя, вы можете быстро переключить балансировщик нагрузки на отправку трафика в синюю среду по умолчанию.

В общем, копируйте, вставляйте и обновляйте ресурсы с меткой `green`. Если вы выражаете ресурсы `blue` в модуле, вам нужно изменить только атрибуты, переданные в модуль. По возможности я разделяю определения зеленой и синей сред в отдельные каталоги или файлы. Это облегчает последующую идентификацию среды.

В файле `shared.py` вы найдете код Python, который упрощает расширение списка сред и среды по умолчанию, прикрепленной к ба-

лансировщику нагрузки. Обычно я определяю список сред и переменную для среды по умолчанию. Затем перебираю список сред и присоединяю атрибуты к балансировщику нагрузки. Это гарантирует, что балансировщик нагрузки высокого уровня может развиваться для адаптации к различным ресурсам и средам.

Добавляя новые ресурсы, вы можете настроить балансировщик нагрузки для отправки трафика в дополнительные среды. Вы можете обновлять IaC вашего балансировщика нагрузки каждый раз, когда захотите запустить blue-green развертывание. Настройка балансировщика нагрузки требует некоторого времени и усилий, но помогает избежать проблем, связанных с изменениями, и контролировать развертывание потенциально опасных обновлений.

9.2.4 *Проведение регрессионного тестирования*

Если вы сразу отправите весь трафик в зеленую сеть и она выйдет из строя, вы можете нарушить работу системы полива росянки. Поэтому вы начинаете с канареечного развертывания и увеличиваете долю трафика в зеленую сеть на 10 % каждый день. Процесс занимает около двух недель, но вы будете уверены, что обновили сеть правильно! Если вы обнаружите проблему, то уменьшите трафик в зеленую сеть и выполните отладку.

На рис. 9.8 показан постепенный процесс увеличения трафика и тестирования «зеленой» среды с течением времени. Вы постепенно уменьшаете долю трафика в синюю среду, пока она не достигнет 0 %, и синхронно увеличиваете долю трафика в зеленую среду, пока она не достигнет 100 %. Вы следите за работой зеленой среды в течение недели или двух, прежде чем отключать синюю сеть на тот случай, если обновление выведет из строя систему в зеленой среде.

Процесс постепенного увеличения трафика и недельного ожидания перед продолжением кажется таким хлопотным! Однако вам необходимо попробовать пропускать достаточно большой трафик через зеленую среду, чтобы определить, можете ли вы продолжить переход на нее. Некоторые сбои сразу проявляют себя при большом трафике через систему, в то время как для обнаружения других требуется время.

Временной интервал, который вы тратите на тестирование, наблюдение и мониторинг системы на предмет ошибок, становится частью регрессионного теста системы. Регрессионное тестирование проверяет, влияют ли изменения в системе на существующие или новые функции. Постепенное увеличение трафика позволяет оценить функциональность системы, одновременно снижая потенциальное влияние сбоя.

ОПРЕДЕЛЕНИЕ *Регрессионный тест* (regression test) проверяет, влияют ли изменения в системе на существующие или новые функциональные возможности.

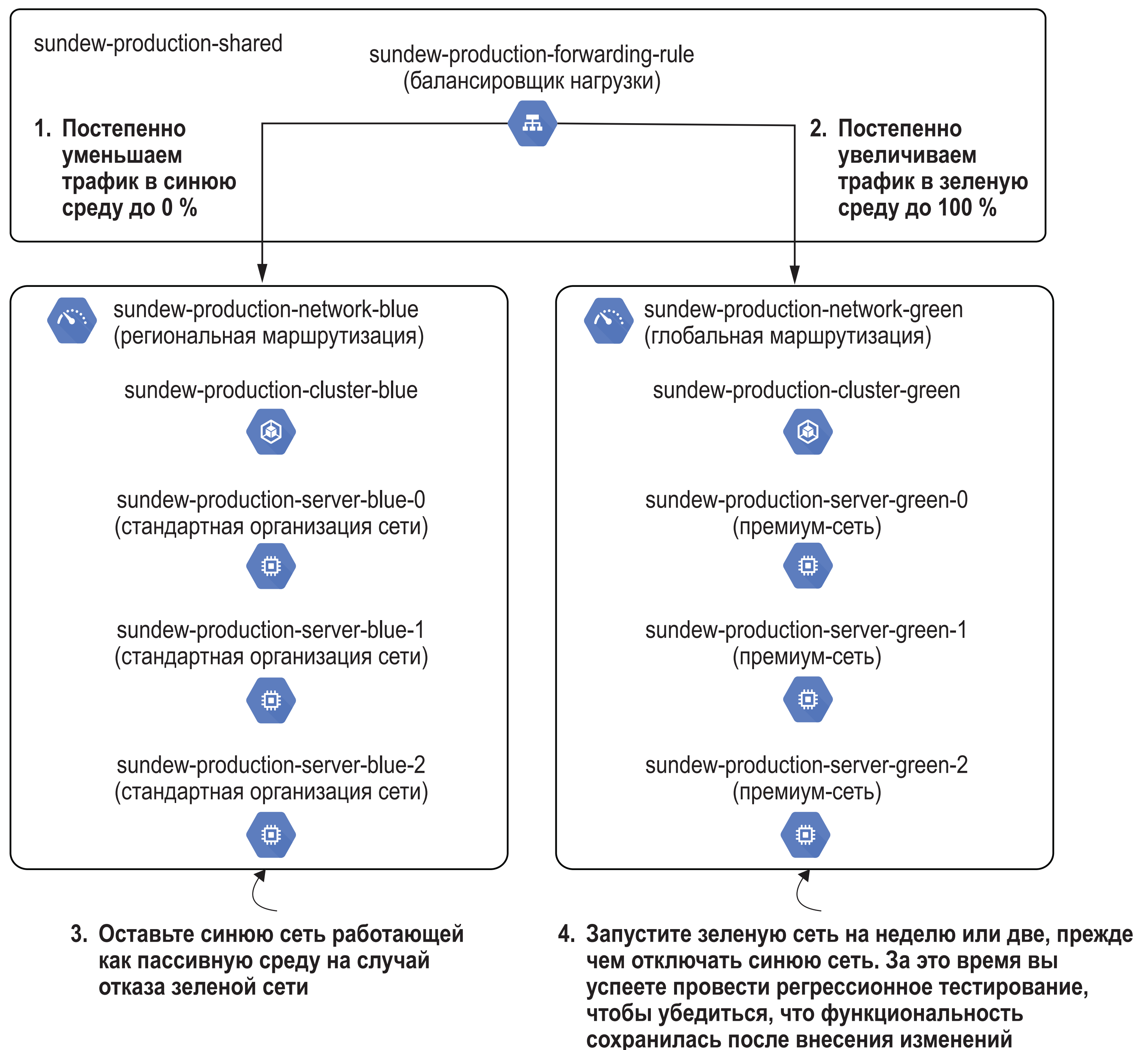


Рис. 9.8 Выделите неделю на регрессионное тестирование, прежде чем направлять весь трафик в новую сеть и удалять старую, что даст вам запас времени для проверки функциональности

С каким шагом следует увеличивать трафик в зеленую среду? Увеличение трафика на 1 % каждый день не даст много информации, если ваша система не обслуживает миллионы запросов. *Постепенность* не предлагает четких критериев, какие приращения следует использовать. Ориентируйтесь по ситуации. Я рекомендую оценить количество запросов, которые ваша система обслуживает ежедневно, и цену сбоя (например, ошибок в запросах пользователей).

Обычно я начинаю с приращения в 10 % и проверяю, сколько запросов это означает для системы. Если я не получаю достаточный размер выборки запросов для выявления сбоев, я увеличиваю приращение. Вставляйте окно регрессионного тестирования между каждым процентным увеличением, чтобы выявить системные сбои.

Даже после достижения отправки всего трафика в зеленую сеть вы все равно должны продолжать запускать тесты и следить за работоспособностью системы в течение недели или двух. Зачем проводить регрессионные тесты в течение нескольких недель? Иногда вы можете столкнуться с пограничными случаями запросов приложений, которые нарушают функциональность. Предоставляя период

для регрессионного тестирования, вы можете наблюдать, сможет ли система обрабатывать неожиданные или необычные полезные нагрузки или запросы.

9.2.5 Удаление синей инфраструктуры

Вы наблюдаете за системой для выращивания росянки в течение двух недель и устраняете все ошибки. Вы знаете, что синяя сеть не обрабатывала ни запросы, ни данные около двух недель, а значит, вы можете удалить ее без дополнительных действий по миграции. Вы подтверждаете бездействие с помощью проверки коллеги или выделенного консультанта. На рис. 9.9 служба по умолчанию обновляется до зеленой среды перед удалением синей среды из IaC.

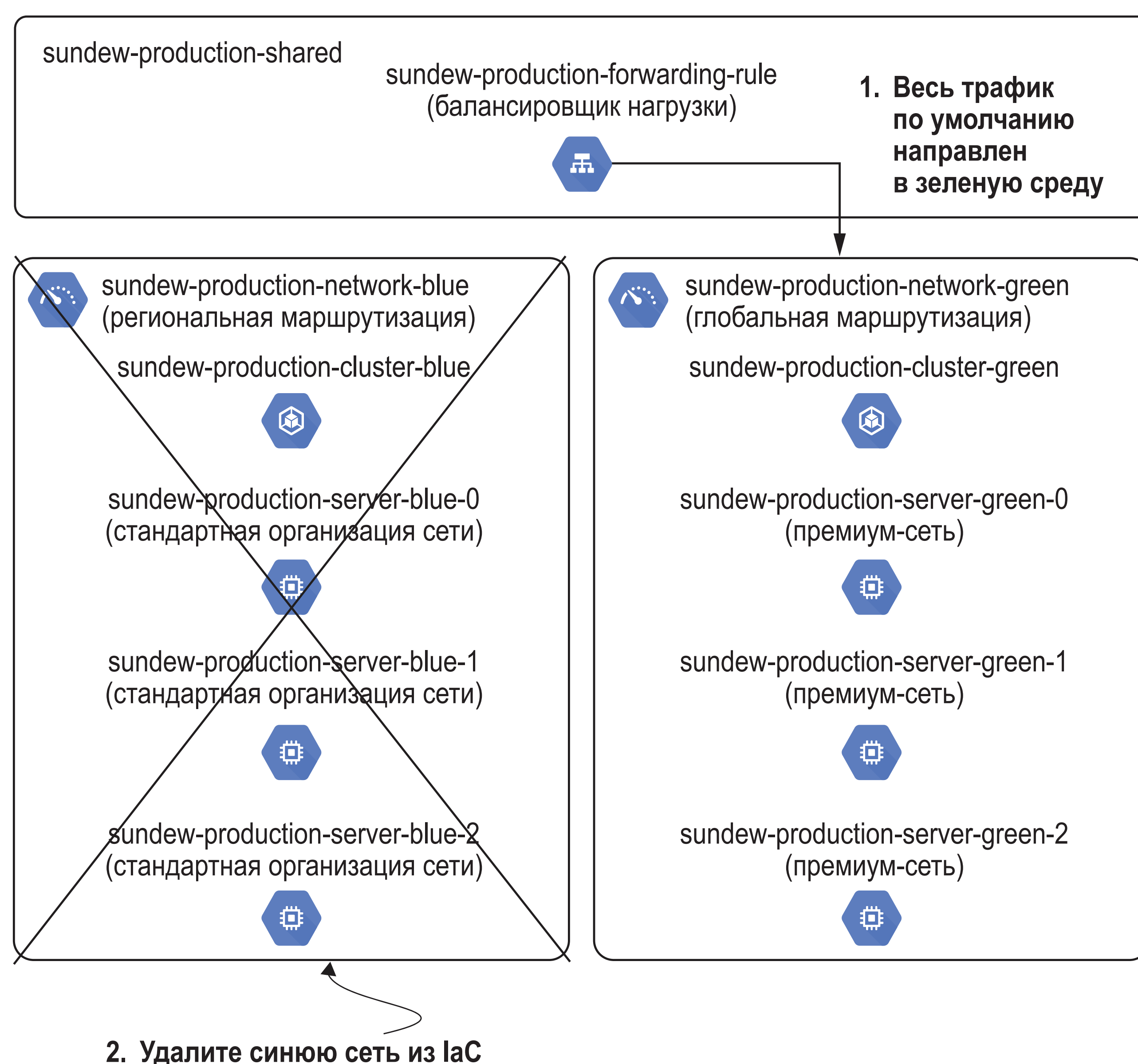


Рис. 9.9 Выведите из эксплуатации старую сеть, удалив ее из IaC и убрав все ссылки

Я считаю удаление синей среды с сетью серьезным изменением, требующим дополнительной экспертной оценки. Вы можете не знать, что кто-то еще использует эту сеть. Ресурсы, которыми вы не делитесь с другими командами, например серверы, могут не нуждаться в дополнительной экспертной проверке или одобрении изменений. Оцените потенциальное влияние удаления среды и классифицируйте изменения на основе паттернов, описанных в главе 7. Давайте настроим балансировщик нагрузки в файле `shared.py`, уста-

новив в качестве службы по умолчанию зеленую сеть и удалив синюю сеть из ее серверных служб, как показано в листинге 9.9.

Листинг 9.9 Удаление балансировщика нагрузки синей среды

```
import blue ❶
import green ❶

TEAM = 'sundew'
ENVIRONMENT = 'production'
PORT = 8080

shared_name = f'{TEAM}-{ENVIRONMENT}-shared'

default_version = 'green' ❷

services_list = [ ❸
    {
        'version': 'green',
        'zone': green.zone,
        'name': f'{shared_name}-green',
        'weight': 100 ❹
    }
]
```

- ❶ Импортируем IaC как для синей, так и для зеленой среды.
- ❷ Меняем версию сети по умолчанию для балансировщика нагрузки на green.
- ❸ Удаляем синюю сеть и экземпляры из списка серверных служб, подлежащих созданию.
- ❹ Отправляем весь трафик в зеленую сеть

Эквиваленты AWS и Azure

Код из листинга 9.9 остается прежним для AWS и Azure. Вам потребуется сопоставить версию, зону доступности, имя и вес с AWS ALB или диспетчером трафика Azure Traffic Manager.

Итак, вы применили изменения к балансировщику нагрузки. Не удаляйте синие ресурсы сразу после этого, поскольку необходимо убедиться, что балансировщик нагрузки не ссылается ни на какие синие ресурсы. После тестирования изменений вы удалите код для создания синей среды из `main.py` и оставите только зеленую среду, как показано в листинге 9.10.

Листинг 9.10 Удаление синей среды из `main.py`

```
import green
import json

if __name__ == "__main__":
```



```
resources = {
    'resource':
        shared.build() +           ❶
        green.build()              ❷
}

with open('main.tf.json', 'w') as outfile: ❸
    json.dump(resources, outfile,          ❸
                  sort_keys=True, indent=4) ❸
```

- ❶ Применяем общий модуль для создания конфигурации JSON глобального балансировщика нагрузки.
- ❷ Применяем зеленый модуль для создания конфигурации JSON сети с глобальной маршрутизацией, серверов с сетью премиум-класса и кластера.
- ❸ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Вы применяете изменения, и ваш инструмент IaC удаляет все синие ресурсы. Вы решаете удалить файл `blue.py`, чтобы никто не мог создавать новые синие ресурсы. Я рекомендую удалить все файлы, которые вы не используете, чтобы в будущем не запутать ваших товарищей по команде. В противном случае у вас может получиться система с излишними ресурсами, которые на самом деле никому не нужны.

Упражнение 9.1

Рассмотрим следующий код:

```
if __name__ == "__main__":
    network.build()
    queue.build(network)
    server.build(network, queue)
    load_balancer.build(server)
    dns.build(load_balancer)
```

Очередь зависит от сети. Сервер зависит от сети и очереди. Как бы вы выполнили blue-green развертывание для обновления очереди с поддержкой SSL?

Ответы на упражнения даны в приложении В.

9.2.6 Несколько примечаний

Представьте, что команде Cap Sundew нужно снова изменить сеть. Вместо создания новой зеленой сети команда может создать новую синюю сеть и повторить процесс развертывания, регрессионного тестирования и удаления! Поскольку старая синяя сеть больше не существует, это обновление не конфликтует с существующей средой.

Как вы называете свои версии или итерации изменений, не имеет значения, пока вы различаете старые и новые ресурсы. В частности,

у вас должна быть возможность выделения двух наборов диапазонов IP-адресов. Вы должны навсегда зарезервировать один для синей сети, а другой – для зеленой. Такое разделение позволит вам гибко вносить изменения с помощью blue-green развертывания без необходимости поиска открытого сетевого пространства.

В целом я выбираю blue-green стратегию развертывания, когда сталкиваюсь со следующими обстоятельствами:

- откат обновлений ресурса занимает много времени;
- я не уверена, что смогу отменить изменения ресурса после развертывания;
- ресурс имеет много зависимостей высокого уровня, которые я не могу надежно определить;
- изменение ресурса затрагивает критически важные приложения, которые не могут простаивать.

Не все ресурсы инфраструктуры должны использовать blue-green развертывание. Например, вы можете обновить действующие политики IAM и быстро отменить их, если обнаружите проблему. Подробнее об отмене изменений вы узнаете в главе 11.

Стратегия blue-green развертывания требует меньше времени и денег, чем поддержка нескольких сред. Однако эта стратегия будет стоить *дороже*, если вам придется развернуть ресурсы инфраструктуры низкого уровня, такие как сети, проекты или учетные записи! Я обычно считаю, что стратегия blue-green развертывания стоит своих денег. Она изолирует изменения в конкретных ресурсах и минимизирует связанные с обновлением риски сбоев в работе производственных систем.

9.3 Инфраструктура, хранящая данные

В примерах этой главы опущен важный класс ресурсов инфраструктуры. Однако система выращивания росянки содержит множество ресурсов, которые обрабатывают, управляют и хранят данные. Например, в состав системы входит база данных Google SQL, работающая в сети с региональной маршрутизацией.

9.3.1 Blue-green развертывание

Члены команды разработчиков приложения для выращивания росянки напоминают вам, что им необходимо обновить свою базу данных, чтобы использовать новую сеть с глобальной маршрутизацией. Вы обновляете идентификатор частной сети в IaC и отправляете изменения в свой репозиторий. Ваш конвейер развертывания не проходит тест на соответствие требованиям (о чем вы узнали в главе 8).

Вы заметили, что тест выполняет пробный прогон на предмет того, не будет ли удалена база данных:


```

$ pytest . -q
F
[100%]
===== FAILURES =====
_____ test_if_plan_deletes_database _____

database = {'address': 'google_sql_database_instance.blue', 'change':
➡{'actions': ['delete'], 'after': None, 'after_sensitive': False,
➡'after_unknown': {}, ...}, 'mode': 'managed', 'name': 'blue', ...}

    def test_if_plan_deletes_database(database):
>         assert database['change']['actions'][0] != 'delete'
E         AssertionError: assert 'delete' != 'delete'

test/test_database_plan.py:35: AssertionError
===== short test summary info =====
FAILED test/test_database_plan.py::test_if_plan_deletes_database -
➡AssertionError: assert 'delete' != 'delete'
1 failed in 0.04s

```

Тест на соответствие не позволяет вам удалить критически важную базу данных! Если бы вы применили изменение без проверки, то удалили бы все данные команды! Вполне естественно, что команду пугает такая перспектива, поэтому вам необходимо выполнить blue-green развертывание.

На рис. 9.10 вы вручную проверяете возможность миграции базы данных в зеленую сеть. Но выясняется, что вы не можете перенести базу данных «как есть». Система выращивания росянки может внести в базу отсутствующие данные, поэтому вы копируете IaC для синей базы данных и создаете новый экземпляр зеленой базы данных в сети премиум-класса. После переноса данных из синей базы данных в зеленую вы переключаете приложения на использование новой базы данных и удаляете старую.



Рис. 9.10 Если вы не можете перенести существующую базу данных, вам необходимо развернуть новую зеленую базу данных, перенести и согласовать данные, а также изменить конечную точку базы данных с синей на зеленую

Стратегия blue-green развертывания применяется к базе данных – классу инфраструктурных ресурсов, хранящих состояние. Ре-

судсы инфраструктуры, хранящие состояние, такие как база данных, хранят данные и управляют ими. Строго говоря, все приложения обрабатывают и хранят определенный объем данных. Но инфраструктура, хранящая состояние, требует особой осторожности, поскольку изменения могут напрямую повлиять на сохраняемые данные. К этому типу инфраструктуры относятся базы данных, очереди, кеши или инструменты потоковой обработки.

ОПРЕДЕЛЕНИЕ В хранящую состояние инфраструктуру входят ресурсы, которые хранят данные и управляют ими.

Почему принципиально важно использовать blue-green развёртывание для ресурсов инфраструктуры, хранящих состояние, т. е. данные? Иногда невозможно обновить ресурс с помощью IaC. Простая замена ресурса может привести к повреждению или потере данных, что повлияет на ваши приложения. Blue-green развёртывание позволяет протестировать функциональность вашей новой базы данных, прежде чем приложения начнут ее использовать.

9.3.2 Конвейер поставки обновлений

Вернемся к команде Cap Sundew. Необходимо исправить конвейер поставки, чтобы автоматизировать обновление базы данных. На рис. 9.11 показано добавление в ваш контейнер нового шага, который автоматически переносит данные из синей базы данных в зеленую. Когда вы добавляете зеленую базу данных и разворачиваете ее, ваш конвейер разворачивает новую базу данных, запускает интег-

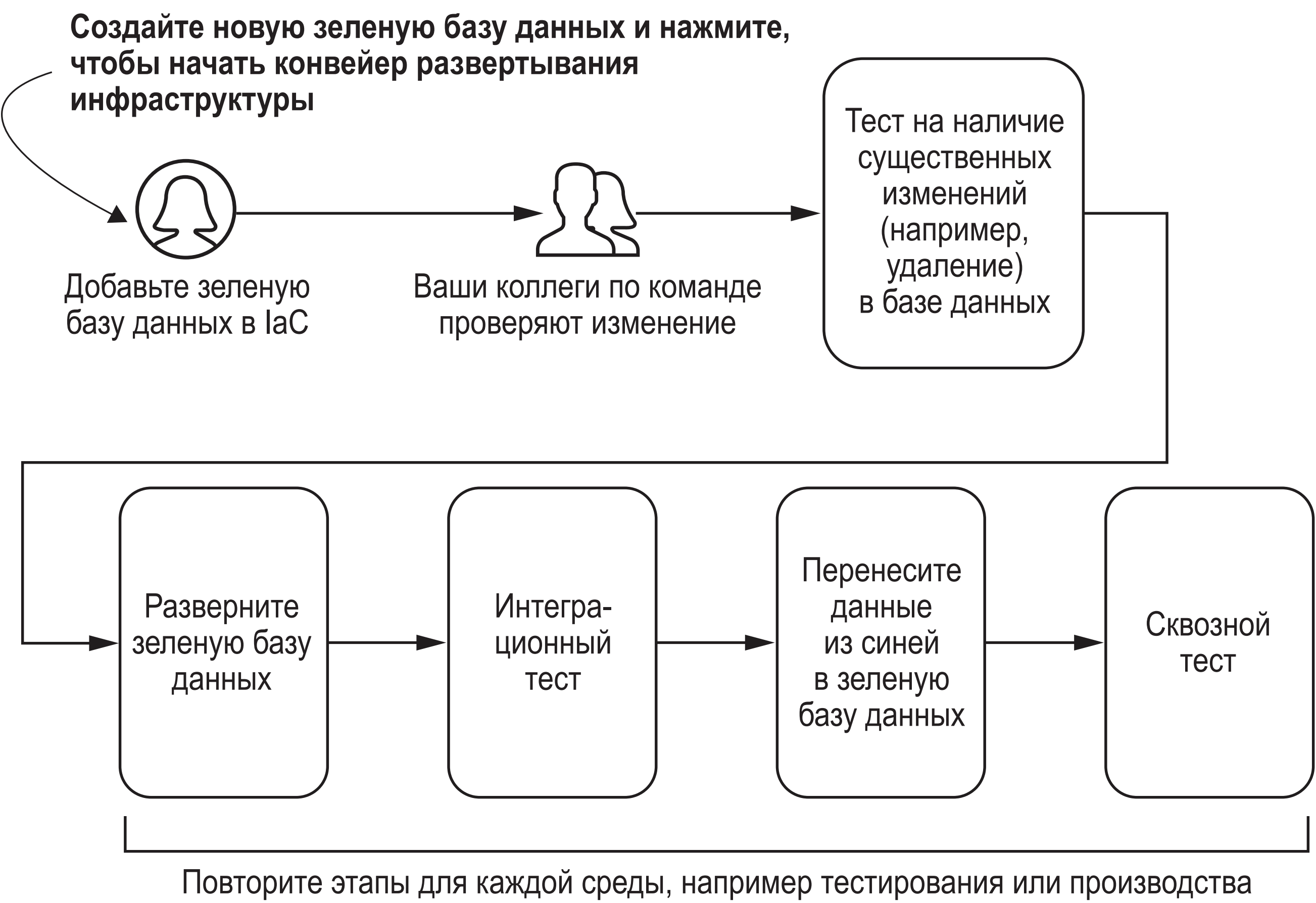


Рис. 9.11 Конвейер развертывания инфраструктуры должен добавить зеленую базу данных и скопировать в нее данные из синей базы

рациональные тесты, автоматически переносит данные из синей базы данных в зеленую и выполняет сквозные тесты.

Сохраните *идемпотентность* с помощью автоматизации этапа миграции. Ваш сценарий миграции при каждом запуске должен приводить к одному и тому же состоянию базы данных. Следует избегать дублирования данных при повторном запуске сценария миграции. Процесс миграции данных различается в зависимости от типа имеющейся у вас инфраструктуры, хранящей состояние (база данных, очередь, кеш или инструменты потоковой обработки).

ПРИМЕЧАНИЕ Вопросам миграции и управления хранящей состояние инфраструктурой с минимальным (или нулевым) временем простоя посвящены целые книги. Для ознакомления с другими паттернами и практиками управления базами данных я рекомендую книгу *Database Reliability Engineering* (Laine Campbell and Charity Majors; O'Reilly, 2017). Вы можете также изучить техническую документацию по миграции, обновлению и доступности других ресурсов инфраструктуры, хранящей состояние.

При обновлении инфраструктуры, хранящей состояние, вместо прямого обновления IaC необходимо запускать автоматический перенос данных в конвейере развертывания. Разделение миграции данных позволяет изменять любые шаги и устранять проблемы с миграцией независимо от создания и удаления ресурсов, хранящих состояние.

9.3.3 Канареечное развертывание

После обновления базы данных для команды Cap Sundew необходимо внести изменения в конфигурацию приложения, чтобы оно могло использовать зеленую базу данных. Для перевода такого высокоуровневого ресурса, как приложение, на новую базу данных можно применить модифицированное канареечное развертывание.

На рис. 9.12 показан паттерн регрессионного тестирования и настройки приложения для использования базы данных. После этапа регрессионного тестирования (чтобы убедиться, что функциональность полностью сохранилась) мы обновляем приложение, чтобы оно *записывало* данные исключительно в зеленую базу данных. После следующего периода регрессионного тестирования мы обновляем приложение для *чтения* данных из зеленой базы.

Итак, вы постепенно внедряете изменения в приложение, чтобы иметь возможность вернуться к синей базе данных, если возникнет проблема. Поскольку приложение записывает данные в две базы, возможно, вам придется разработать дополнительную автоматизацию для согласования данных. Но только запись в новую инфраструктуру, хранящую состояние, гарантирует тестирование всех

критически важных функций, связанных с хранением и обновле-
нием данных. Только тогда можно ожидать, что приложения смогут
правильно читать и обрабатывать данные.



Рис. 9.12 Мы постепенно обновляем конвейер развертывания приложения для записи в обе базы данных, записи только в зеленую базу данных, а затем для чтения из зеленой базы

Теперь, когда приложения для выращивания росянки используют зеленую базу данных, синяя база данных удаляется из IaC, как по-
казано на рис. 9.13. Обратите внимание, что тесты на соответствие не пройдут, потому что вы хотите удалить базу данных! Вы провали-
те тест, если планируете удалить зеленую базу данных, а не синюю. Ручное переопределение позволяет удалить синюю базу данных, по-
скольку приложения больше не используют ее.

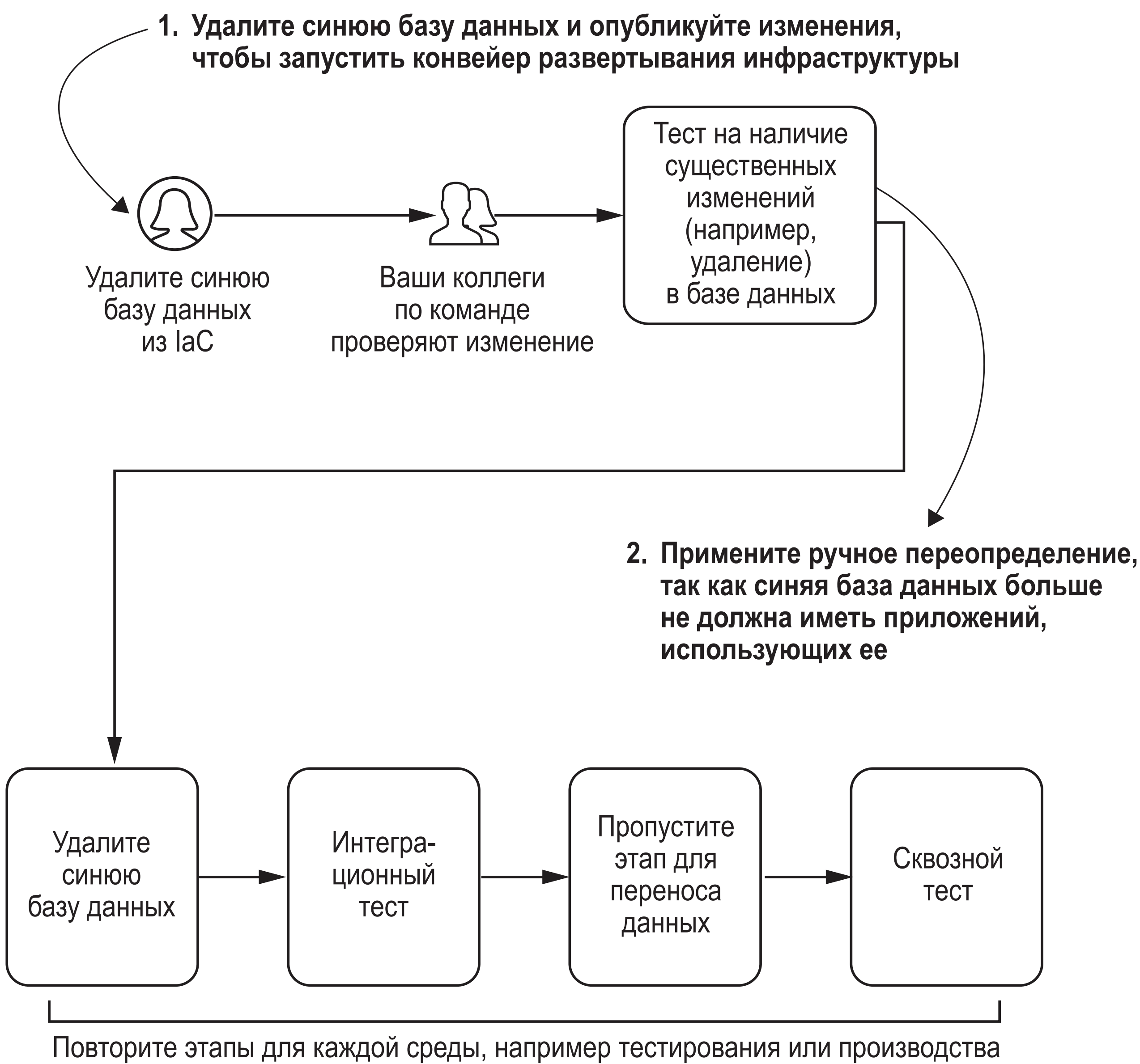


Рис. 9.13 Удалите синюю базу данных, убрав ее из IaC и отправив изменения в конвейер развертывания

Такие методы, как канареечное развертывание, обеспечивают быструю обратную связь, чтобы смягчить последствия сбоя, особенно в ситуациях, связанных с обработкой данных. Между исправлением нескольких неправильных записей в базе данных и ее полным восстановлением из резервной копии есть большая разница! Мне намного комфортнее, когда я знаю, что вношу изменения в инфраструктуру, сохраняющую состояние, в изолированной зеленой среде, а не в работающей производственной системе.

Стратегии, основанные на неизменности, такие как blue-green развертывание, предусматривают структурированный процесс внесения изменений и минимизацию потенциального ущерба. Благодаря принципу воспроизводимости обычно удается реализовать и принцип неизменности – выполнять обновления IaC путем дублирования и редактирования конфигурации. Этот принцип также позволяет улучшить резервирование инфраструктурной системы с помощью аналогичного процесса дублирования.

Краткое содержание главы

- Прежде чем приступать к каким-либо обновлениям инфраструктуры, убедитесь, что в вашей системе реализованы тестирование, мониторинг и наблюдаемость инфраструктуры и приложений.
- Избыточность в IaC означает добавление в конфигурацию дополнительных простаивающих ресурсов, чтобы можно было выполнить аварийное переключение в случае сбоя компонента.
- Добавление резервной конфигурации в IaC может повысить надежность системы – показатель того, насколько долго система работает правильно в течение определенного периода времени.
- Комбинированная конфигурация состоит из активной среды, обслуживающей запросы, и дублирующей резервной среды на случай сбоя системы.
- При аварийном переключении трафик перенаправляется из неисправной активной среды в пассивную среду.
- Конфигурация «активный–активный» задает две активные среды, обслуживающие запросы; обе среды можно копировать и управлять ими с помощью IaC.
- Blue-green развертывание создает новое подмножество ресурсов инфраструктуры, которое содержит необходимые изменения и постепенно принимает на себя все запросы. Затем вы можете удалить старый набор ресурсов.
- Согласно стратегии blue-green развертывания создайте ресурс, который вы хотите изменить, и ресурсы высокого уровня, которые от него зависят.
- При канареечном развертывании небольшой процент трафика направляется на новые ресурсы инфраструктуры, чтобы прове-

рить правильность работы системы. Постепенно вы увеличиваете долю трафика.

- Выделите несколько недель на регрессионное тестирование, чтобы проверить, влияют ли изменения в системе на существующие или новые функции.
- Ресурсы инфраструктуры, хранящие состояние, такие как базы данных, кеши, очереди и инструменты потоковой обработки, хранят данные и управляют ими.
- Добавьте этап миграции к конвейеру blue-green развертывания ресурсов инфраструктуры, хранящей состояние. Вам необходимо копировать данные между синей и зеленой инфраструктурами, хранящими состояние.

10

Рефакторинг

Основные темы главы:

- как определить, когда следует провести рефакторинг IaC, чтобы избежать влияния на систему;
- применение флагов функций для изменения атрибутов инфраструктуры;
- применение последовательного обновления (rolling update) без влияния на работу системы.

Со временем вы можете перерасти шаблоны и методы, которые вы используете для совместной работы над инфраструктурой в виде кода. Даже такие методы внесения изменений, как blue-green развертывание, не всегда могут разрешить конфликты в конфигурации или изменениях, когда ваша команда работает над какой-либо IaC. Вам необходимо внести ряд серьезных изменений в свой IaC и решить проблемы с масштабированием на практике.

Например, команда Cap Sundew по выращиванию росянки из Datacenter for Carnivorous Plants заявляет, что больше не может комфортно и уверенно внедрять новые изменения в свою систему. Команда помещает все ресурсы инфраструктуры в один репозиторий (в соответствии с паттерном «одиночка») и просто продолжает добавлять новые обновления поверх него.

Команда Cap Sundew отмечает несколько проблем в своей системе. Во-первых, команда обнаружила, что обновления конфигурации инфраструктуры постоянно *пересекаются* (конфликтуют). Один член команды работает над обновлением серверов, но обнаруживает, что его коллега по команде обновил сеть и повлиял на настройки серверов.

Во-вторых, выполнение одного изменения занимает более 30 минут. Одно изменение вызывает сотни вызовов API инфраструктуры для получения состояния ресурсов, что замедляет цикл обратной связи.

Наконец, команда безопасности выражает обеспокоенность тем, что инфраструктура для выращивания росянки может иметь небезопасную конфигурацию. Текущая конфигурация не использует утвержденные в компании настройки безопасности инфраструктуры.

Вы приходите к выводу, что нужно изменить конфигурацию системы. В конфигурации должен использоваться набор компонентов и настроек, одобренный командой безопасности. Вам также необходимо разбить конфигурацию на отдельные ресурсы, чтобы минимизировать потенциальный ущерб от неудачных изменений. В этой главе обсуждаются некоторые паттерны и методы IaC для разбиения больших репозиториях с сотнями ресурсов. В качестве помощника по IaC в команде Cap Sundew вы проведете рефакторинг монолитной конфигурации системы в отдельные репозитории и структурируете конфигурации сервера для использования модулей, чтобы избежать конфликтов и обеспечить соответствие требованиям стандартов безопасности.

ПРИМЕЧАНИЕ Для демонстрации рефакторинга требуется достаточно большой (и сложный) пример. Если вы запустите полный пример, вы понесете расходы, превышающие уровень бесплатного пользования GCP. В примеры этой книги включены только самые необходимые строки кода, а остальные опущены для удобства чтения. Полные листинги можно найти в репозитории кода книги по адресу <https://github.com/joatmon08/manning-book/tree/main/ch10>. Если вы адаптируете эти примеры для AWS и Azure, вы также понесете определенные расходы. Когда это возможно, я даю рекомендации по преобразованию используемых примеров в тот облачный сервис, который вы выберете.

10.1 Минимизация последствий рефакторинга

Команде Cap Sundew нужна помощь в перестройке конфигурации инфраструктуры. Вы принимаете решение провести рефакторинг IaC, чтобы лучше изолировать конфликты, сократить время на применение изменений и обеспечить их безопасность в соответствии

со стандартами компании. Рефакторинг IaC заключается в реструктуризации конфигурации или кода и не затрагивает существующие ресурсы инфраструктуры.

ОПРЕДЕЛЕНИЕ *Рефакторинг IaC* – это внесение структурных изменений в конфигурацию или код без воздействия на существующие ресурсы инфраструктуры.

Вы сообщаете команде, что для устранения проблем с инфраструктурой необходимо выполнить рефакторинг конфигурации. Хотя члены команды поддерживают вас, их беспокоит риск сбоя системы после рефакторинга. Вызов принят: вы применяете несколько техник для уменьшения потенциального ущерба при рефакторинге IaC.

Технический долг

Рефакторинг часто бывает направлен на ликвидацию *технического долга*. Понятие технического долга возникло как метафора, обозначающая накопленные в программном коде или архитектуре проблемы, которые затрудняют изменение или расширение всей системы.

Чтобы понять, что означает технический долг применительно к IaC, вспомните, что команда Cap Sundew поместила код, описывающий инфраструктурные ресурсы, в один репозиторий. Команда накапливает долг в виде затрат времени и усилий. Изменение сервера, которое должно занять один день, теперь занимает четыре дня, поскольку команде приходится разрешать конфликты изменений и ждать обработку сотни запросов к API инфраструктуры. Имейте в виду, что в сложных системах всегда присутствует некоторый технический долг и вам придется постоянно прикладывать усилия по его минимизации.

Руководство компаний не любит, когда говорят, что нужно устранить технический долг, потому что это не связано напрямую с операционной деятельностью. Я утверждаю, что технический долг, который вы накапливаете в инфраструктуре, всегда будет преследовать вас. Демон технического долга пробуждается, когда кто-то меняет инфраструктуру и вызывает простои приложений или, что еще хуже, нарушение безопасности, которое раскрывает личную информацию и влечет за собой денежные затраты. Оценка последствий застарелого технического долга помогает оправдать затраты перед руководством.

10.1.1 Уменьшение потенциального ущерба с помощью стратегии последовательного обновления (*rolling update*)

Платформа Datacenter for Carnivorous Plants и команда безопасности предлагают серверный модуль настроек безопасности, который вы можете использовать для системы выращивания росянки. Инфра-

структура команды Cap Sundew содержит три конфигурации серверов, но без использования этого модуля. Как изменить IaC для использования модуля настроек безопасности.

Представьте, что вы создаете три новых сервера одновременно и сразу отправляете на них трафик. Если приложения на сервере работают некорректно, вы можете полностью нарушить работу системы полива росянки, и бедные растения засохнут! Более разумно было бы уменьшить охват изменений, постепенно меняя серверы по одному.

На рис. 10.1 показано, как вы создаете одну конфигурацию сервера с помощью модуля, развертываете приложение на новом сервере, проверяете работу приложения, затем удаляете старый сервер. Вы повторяете процесс еще два раза для каждого сервера. Вы полностью развертываете изменение на одном сервере, прежде чем обновлять следующий.

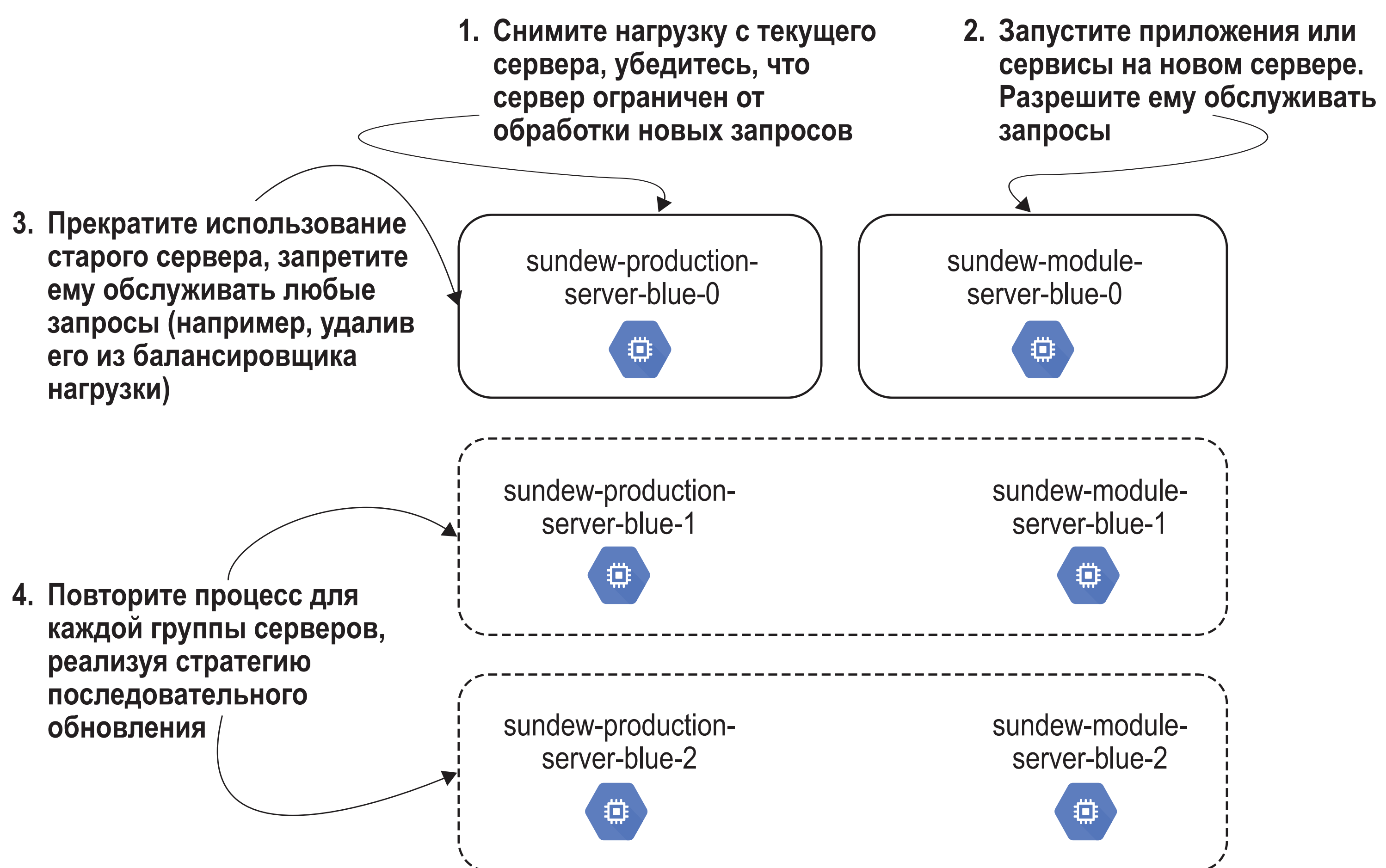


Рис. 10.1 Последовательное обновление применяется для создания нового сервера, развертывания приложения и тестирования его функциональности и сводит к минимуму сбои в работе других ресурсов

При использовании стратегии *последовательного обновления* однотипные ресурсы постепенно заменяются новыми экземплярами и проходят тестирование, прежде чем обновление продолжится на следующих экземплярах.

ОПРЕДЕЛЕНИЕ *Последовательное обновление* (rolling update) – это практика изменения группы похожих ресурсов последовательным замещением изменениями и их тестирования перед внедрением изменения в следующий ресурс.

Последовательное обновление конфигурации команды Cap Sundew изолирует сбои на отдельных серверах каждый раз, когда вы их об-

новляете, и позволяет вам протестировать функциональность обновленного сервера, прежде чем приступить к следующему обновлению.

Практика последовательного обновления может избавить вас от необходимости распутывать большой набор неудачных изменений или неправильных настроек IaC. Например, если модуль подразделения Datacenter for Carnivorous Plants не работает на первом сервере, значит, его не следует разворачивать на остальных серверах. Механизм последовательного обновления позволяет вам проверить совместимость с текущей IaC для каждого сервера, прежде чем переходить к следующему. Постепенный подход также снижает вероятность простоя приложений или сбоев при обновлении серверов.

ПРИМЕЧАНИЕ Я заимствовала термин «последовательное обновление» у платформ управления контейнеризацией, таких как Kubernetes (k8s). Когда вы захотите обновить узлы (виртуальные машины) оркестратора рабочей нагрузки, то обнаружите, что он использует механизм автоматического последовательного обновления. Оркестратор (k8s) блокирует старый узел, предотвращает запуск на нем новых рабочих нагрузок, запускает нужные процессы на новом узле, останавливает все процессы на старом узле и отправляет рабочие нагрузки и запросы на новый узел. Ваша задача – имитировать этот процесс при рефакторинге!

Благодаря стратегии последовательного обновления и дополнительному тестированию вы будете уверены, что серверы смогут работать с новыми настройками политик безопасности. Вы сообщаете команде, что завершили рефакторинг серверов и убедились, что они работают с внутренними сервисами. Команда Cap Sundew теперь может отправлять весь клиентский трафик на новые защищенные серверы. Однако члены команды говорят вам, что сначала им необходимо обновить балансировщик клиентской нагрузки!

10.1.2 Поэтапный рефакторинг с использованием флагов функций (feature flags)

Вам нужен способ скрыть новые серверы от балансировщика нагрузки на несколько дней и подключить их, когда команда даст разрешение. Но у вас уже готовы все конфигурации! Вам будет достаточно скрыть новые серверы с помощью одной переменной, чтобы упростить работу команды Cap Sundew. Когда члены команды завершат обновление балансировщика нагрузки, они обновят переменную, чтобы подключить к нему новые серверы.

На рис. 10.2 показано, как оперировать переменной для добавления новых серверов, созданных модулем. Мы создаем логическую переменную для включения или отключения нового серверного

модуля, которая может принимать значение `True` или `False`. Затем мы добавляем в IaC оператор `if`, который ссылается на логическое значение. Значение `True` добавляет новые серверы в балансировщик нагрузки. Значение `False` удаляет серверы из балансировщика нагрузки.

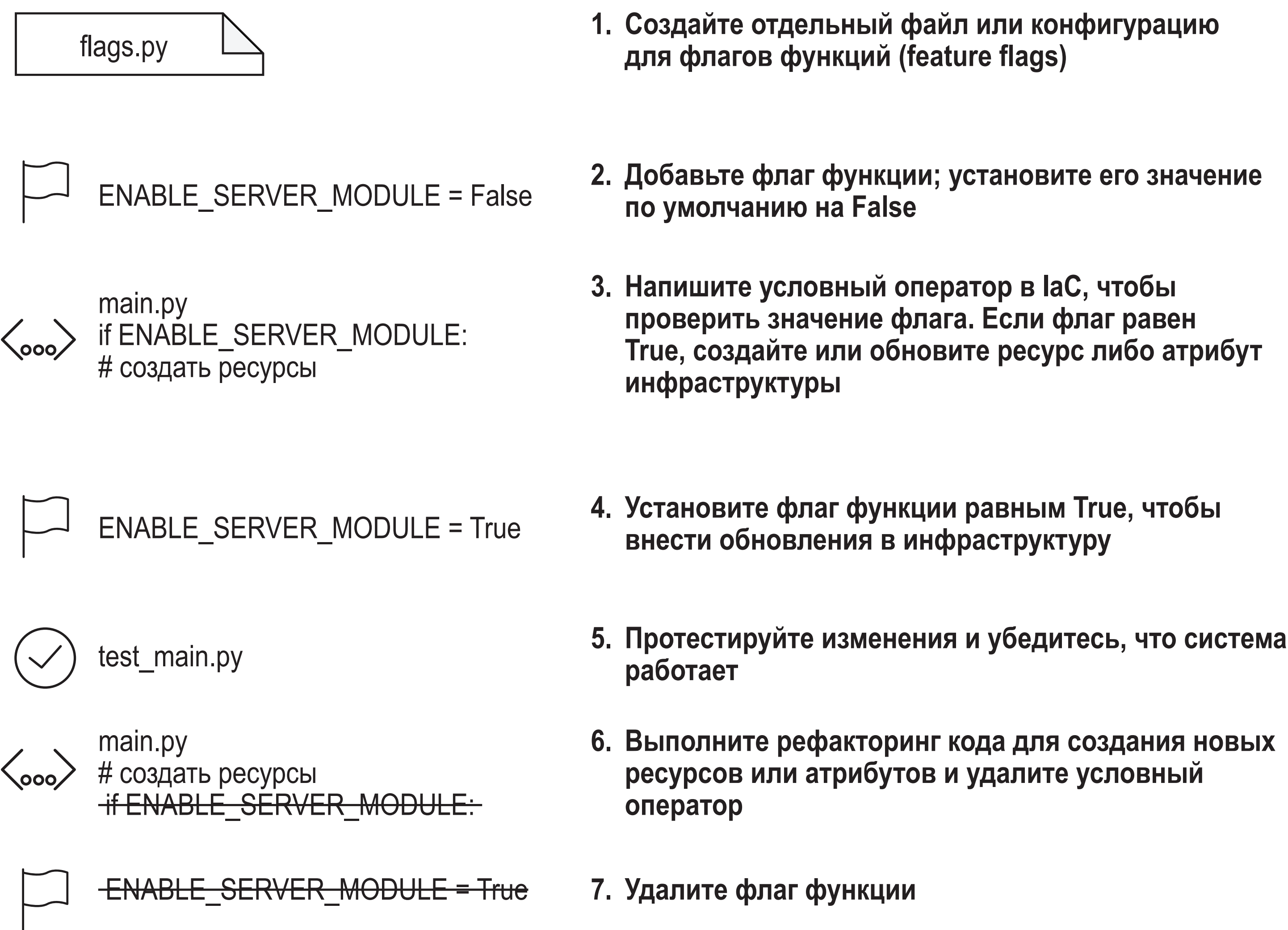


Рис. 10.2 Флаги функций в IaC затрагивают процессы создания, управления и удаления

Логическая переменная обеспечивает соблюдение принципов компонентности и развиваемости IaC. Одно изменение переменной добавляет, удаляет или обновляет конфигурацию. Переменная, называемая *флагом функции* (или переключателем функции), используется для включения или отключения ресурсов инфраструктуры, зависимостей и атрибутов. Флаги функций часто встречаются при разработке программного обеспечения с использованием магистральной модели разработки (они работают только в главной ветке).

ОПРЕДЕЛЕНИЕ Флаги функций (feature flag, переключатель функции) включают или отключают ресурсы инфраструктуры, зависимости или атрибуты с помощью логического значения.

Флаги скрывают определенные функции или код и не позволяют им влиять на работу остальной части команды в главной ветке. По просьбе команды Cap Sundew вы скрываете новые серверы от балансировщика нагрузки до тех пор, пока команда не завершит обнов-

ление балансировщика нагрузки. Аналогичным образом вы можете использовать флаги функций в IaC для поэтапной настройки и отправки обновления с помощью одной переменной.

Добавление флага

Чтобы реализовать механизм флагов функций и поэтапного обновления новых серверов, вы добавляете флаг и устанавливаете его в значение `False`. Вы устанавливаете флаг функции по умолчанию в значение `False`, чтобы сохранить исходное состояние инфраструктуры, как показано на рис. 10.3. Конфигурация системы выращивания росянки по умолчанию отключает модуль сервера, так что с исходными серверами ничего не происходит.

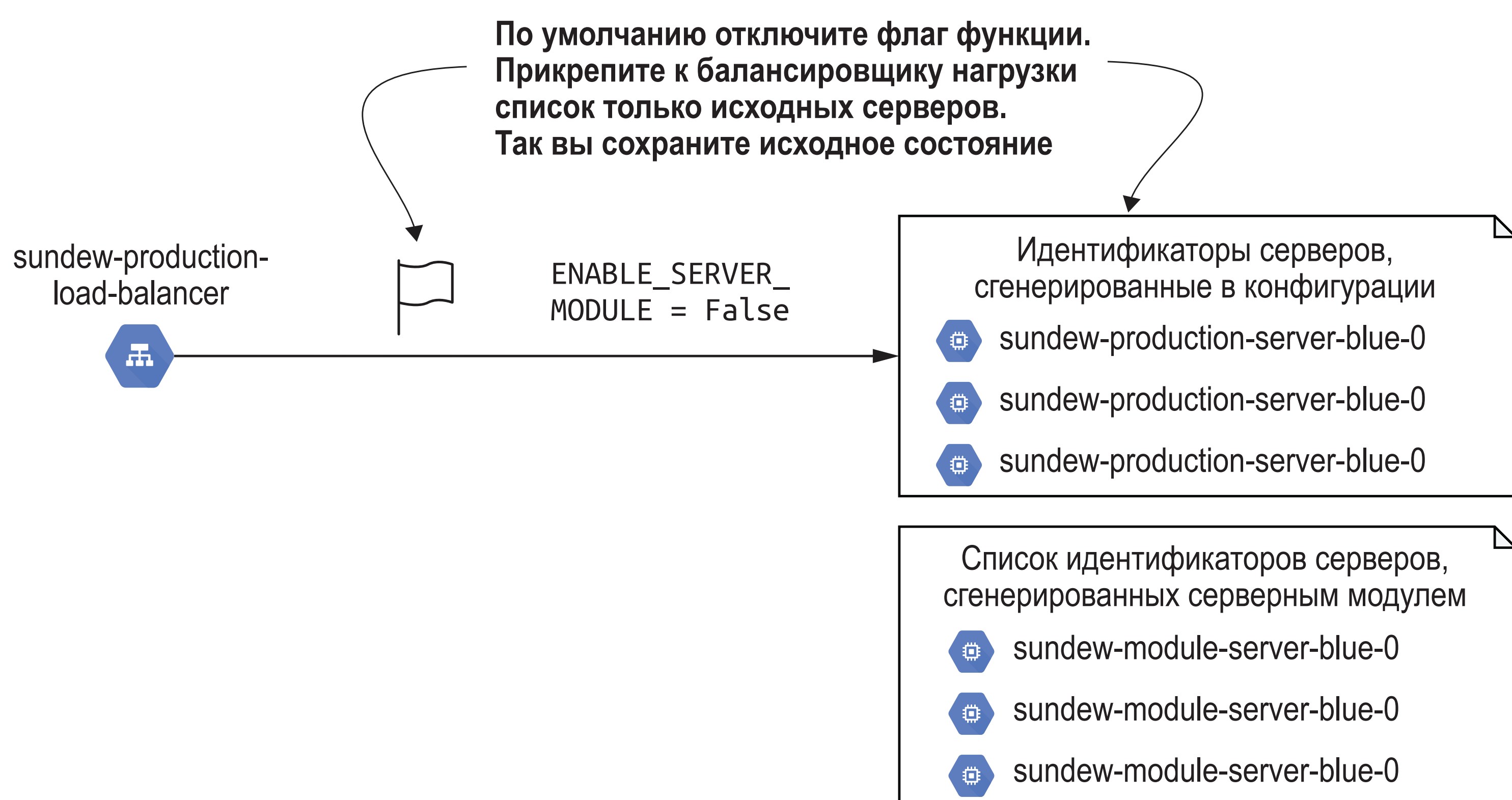


Рис. 10.3 Установите для флага функции значение `False` по умолчанию, чтобы сохранить исходное состояние и зависимости ресурсов инфраструктуры

Давайте реализуем флаг функции в коде Python. Установим флаг модуля сервера по умолчанию на `False` в отдельном файле с именем `flags.py`. Файл определяет флаг `ENABLE_SERVER_MODULE` и задает ему значение `False`:

```
ENABLE_SERVER_MODULE = False
```

Вы также можете вставлять флаги функций в качестве переменных в другие файлы, но так их можно потерять из виду! Лучше поместить их в отдельный файл Python.

ПРИМЕЧАНИЕ Я всегда определяю флаги функций в файле, чтобы находить и изменять их в одном месте.

В листинге 10.1 мы импортируем флаг функции в `main.py` и добавляем логику для создания списка серверов, подлежащих подключению к балансировщику нагрузки.

Листинг 10.1 Использование флага функции для добавления серверов в балансировщик нагрузки

```
import flags 1
def _generate_servers(version):
    instances = [ 2
        f'${{google_compute_instance.{version}_0.id}}', 2
        f'${{google_compute_instance.{version}_1.id}}', 2
        f'${{google_compute_instance.{version}_2.id}}' 2
    ] 2
    if flags.ENABLE_SERVER_MODULE: 3
        instances = [ 4
            f'${{google_compute_instance.module_{version}_0.id}}', 4
            f'${{google_compute_instance.module_{version}_1.id}}', 4
            f'${{google_compute_instance.module_{version}_2.id}}', 4
        ]
    return instances
```

- ❶ Импортируем файл, который определяет все флаги функций.
- ❷ Определяем список существующих вычислительных экземпляров (серверов) Google с помощью ресурса Terraform в системе.
- ❸ Используем условный оператор для оценки флага функции и добавляем ресурсы серверного модуля в балансировщик нагрузки.
- ❹ Если для флага функции установлено значение True, серверы, созданные модулем, будут подключены к балансировщику нагрузки. В противном случае исходные серверы останутся.

Эквиваленты AWS и Azure

Чтобы перенести код листинга 10.1 в AWS или Azure, используйте ресурс Terraform AWS EC2 (<http://mng.bz/VMMr>) или ресурс Terraform виртуальной машины Azure Linux (<http://mng.bz/xnnq>). Вам нужно будет обновить только ссылки в списке экземпляров.

Запустите Python с отключенным флагом функции, чтобы сгенерировать конфигурацию JSON. Полученная конфигурация JSON (листинг 10.2) добавляет в балансировщик нагрузки только исходные серверы, что сохраняет существующее состояние ресурсов инфраструктуры.

Листинг 10.2 Конфигурация JSON с отключенным флагом функции

```
{
  "resource": [
    {
      "google_compute_instance_group": {
        "blue": [
          {
            "instances": [
              1
```



```

        "${google_compute_instance.blue_0.id}", ❷
        "${google_compute_instance.blue_1.id}", ❷
        "${google_compute_instance.blue_2.id}" ❷
    ]
}
]
}
}
]
}

```

- ❶ Создаем группу вычислительных экземпляров (серверов) Google, используя ресурс Terraform для подключения к балансировщику нагрузки.
- ❷ Конфигурация содержит список исходных вычислительных экземпляров Google, сохраняя текущее состояние ресурсов инфраструктуры.

Эквиваленты AWS и Azure

Группа экземпляров Google Compute не имеет прямого эквивалента в AWS или Azure. Вместо этого вам потребуется заменить группу вычислительных экземпляров определением ресурса целевой группы AWS (<http://mng.bz/AyyE>) для балансировщика нагрузки. Для Azure вам понадобится пул внутренних адресов и три адреса экземпляров виртуальных машин (<http://mng.bz/ZAAj>).

Установка флага функции по умолчанию в значение `False` соответствует принципу идемпотентности. Когда вы запускаете сценарий IaC, состояние вашей инфраструктуры не должно меняться. Значение флага по умолчанию гарантирует, что вы случайно не измените существующую инфраструктуру. Сохранение исходного состояния существующих серверов сводит к минимуму сбои в работе зависимых приложений.

Включение флага

Команда Cap Sundew внесла свои изменения и дала согласие на добавление новых серверов, созданных модулем, в балансировщик нагрузки. Вы устанавливаете для флага функции значение `True`, как показано на рис. 10.4. При развертывании обновления вы подключаете серверы из модуля к балансировщику нагрузки и удаляете старые серверы.

Рассмотрим обновленный флаг функции в действии. Начнем с установки флага функции для серверов в значение `True`:

```
ENABLE_SERVER_MODULE = True
```

Далее выполняем код Python, чтобы сгенерировать новую конфигурацию JSON. Конфигурация в листинге 10.3 теперь содержит серверы, созданные модулем, который вы подключите к балансировщику нагрузки.

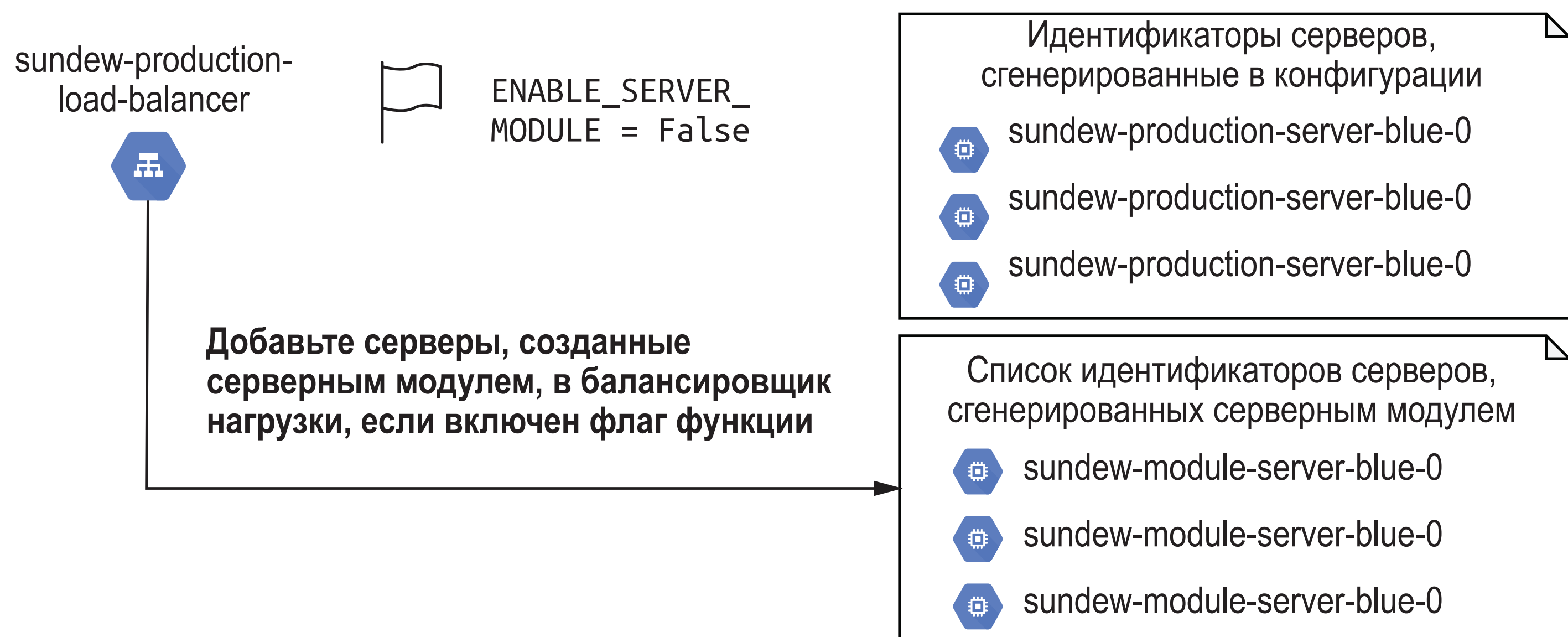


Рис. 10.4 Задаем для флага функции значение True, подключаем три новых сервера, созданных модулем, к балансировщику нагрузки и отсоединяем старые серверы

Листинг 10.3 Конфигурация JSON с включенным флагом функции (feature flags)

```
{
  "resource": [
    {
      "google_compute_instance_group": {
        "blue": [
          {
            "instances": [
              "${google_compute_instance.module_blue_0.id}", ❶
              "${google_compute_instance.module_blue_1.id}", ❶
              "${google_compute_instance.module_blue_2.id}" ❶
            ]
          }
        ]
      }
    }
  ]
}
```

- ❶ Новые серверы, созданные модулем, заменяют старые серверы, поскольку вы включили флаг функции.

Флаг функции позволяет размещать низкоуровневые серверные ресурсы модуля, не затрагивая высокоуровневую зависимость балансировщика нагрузки. Вы можете повторно запустить код с отключенной функцией, чтобы вернуть старые серверы.

Зачем использовать флаг функции для переключения на новый серверный модуль? Флаг функции скрывает функциональность новых серверов от рабочей нагрузки до тех пор, пока мы не почувствуем, что готовы развернуть связанные с ней ресурсы. Таким образом, нам достаточно одной переменной для добавления, удаления или обновления набора ресурсов. Эту же переменную можно использовать для отмены обновлений.

Удаление флага

После проверки работоспособности серверов в течение некоторого времени команда сообщает, что к новому серверному модулю нет замечаний. Теперь вы можете удалить старые серверы при помощи кода из листинга 10.4. Вам больше не нужен флаг функции, и вы не хотите запутать других членов команды, когда они будут читать код. Вы выполняете рефакторинг кода Python для балансировщика нагрузки, чтобы удалить старые серверы и флаг функции.

Листинг 10.4 Удаление флага функции после завершения изменения

```
import blue ❶

def _generate_servers(version):
    instances = [ ❷
        f'${{google_compute_instance.module_{version}_0.id}}', ❷
        f'${{google_compute_instance.module_{version}_1.id}}', ❷
        f'${{google_compute_instance.module_{version}_2.id}}', ❷
    ]
    return instances
```

- ❶ Вы можете удалить импорт флагов функций, поскольку они больше не нужны для ваших серверов.
- ❷ Окончательно присоединяем серверы, созданные модулем, к балансировщику нагрузки и удаляем флаг функции.

Предметно-ориентированные языки

В листинге 10.4 показан флаг функции на стандартном языке программирования. Вы можете использовать флаги функций в DSL, но вам придется адаптировать код в соответствии с синтаксисом вашего инструмента. В Terraform вы можете имитировать флаг функции, используя переменную и метааргумент count (<http://mng.bz/R44n>):

```
variable "enable_server_module" {
    type = bool
    default = false
    description = "Задайте True для создания сервера при помощи модуля."
}
module "server" {
    count = var.enable_server_module ? 1 : 0
    ## сокращено для ясности
}
```

В AWS CloudFormation вы можете передать параметр и установить условие (<http://mng.bz/2nnN>), чтобы включить или отключить создание ресурсов:

```
AWSTemplateFormatVersion: 2010-09-09
Description: Truncated example for CloudFormation feature flag
Parameters:
```



```
EnableServerModule:
  AllowedValues:
    - 'true'
    - 'false'
  Default: 'false'
  Description: Задайте True для создания сервера при помощи модуля.
  Type: String
Conditions:
  EnableServerModule: !Equals
    - !Ref EnableServerModule
    - true
Resources:
  ServerModule:
    Type: AWS::CloudFormation::Stack
    Condition: EnableServerModule
  ## Сокращено для ясности
```

Помимо флагов функций для включения и отключения целых ресурсов, вы можете использовать условные операторы для включения или отключения определенных атрибутов ресурса.

Возьмите за правило *удалять* флаг функции после завершения изменения. Избыток флагов функций может загромадить ваш IaC сложной логикой, что затруднит устранение неполадок в конфигурации инфраструктуры.

Варианты применения флагов

В приведенном выше примере флаги функций задействованы в *преобразовании одинарных конфигураций в модульные*. Я часто применяю в таких случаях флаги функций, чтобы упростить создание и удаление ресурсов инфраструктуры. Среди других вариантов использования флагов можно отметить следующие:

- совместная работа и избежание конфликтов изменений одних и тех же ресурсов инфраструктуры или зависимостей;
- создание группы изменений и быстрое их развертывание с помощью одного переключения флага;
- тестирование изменения и быстрое его отключение в случае сбоя.

На применении флагов функций основан метод, позволяющий скрыть или изолировать изменения ресурсов, атрибутов и зависимостей инфраструктуры в процессе рефакторинга конфигурации. Однако переключение флага все равно может нарушить работу системы. В примере с серверами команды Cap Sundew мы не можем просто переключить флаг функции на True и ожидать, что серверы запустят приложение. Чтобы минимизировать сбои в работе системы, необходимо объединить флаг функции с другими методами, такими как последовательное обновление.

10.2 Декомпозиция монолитных конфигураций

Члены команды по выращиванию росянки заявляют, что у них остались проблемы с системой. Вы приходите к выводу, что причина проблем – в монолитной конфигурации с сотнями ресурсов и атрибутов. Всякий раз, когда кто-то вносит изменения, коллеги по команде вынуждены разрешать конфликты со своими настройками конфигурации. Им также приходится ждать примерно 30 минут, чтобы внести свои изменения.

Монолитная архитектура IaC означает определение всех ресурсов инфраструктуры в одном месте. Вам необходимо разбить монолит IaC на более мелкие модульные компоненты, чтобы свести к минимуму рабочие конфликты между членами команды и ускорить развертывание изменений.

ОПРЕДЕЛЕНИЕ *Монолитная архитектура IaC задает все ресурсы инфраструктуры в одной конфигурации и одном и том же состоянии.*

В этом разделе мы рассмотрим рефакторинг монолитной инфраструктуры команды по выращиванию росянки. Самый важный первый шаг заключается в нахождении и группировке ресурсов и зависимостей инфраструктуры высокого уровня. Рефакторинг завершается группировкой инфраструктурных ресурсов низкого уровня.

Монолит или монорепозиторий?

Напомню, что вы можете поместить конфигурацию своей инфраструктуры в единый репозиторий (глава 5). Означает ли наличие единого репозитория монолитную архитектуру? Не обязательно. Вы можете разделить репозиторий на отдельные подкаталоги. Каждый подкаталог содержит отдельную конфигурацию IaC.

Монолитная архитектура означает, что вы управляете многими ресурсами вместе и тесно связываете их, что затрудняет изолированное изменение подмножества. Монолит обычно возникает из исходного одиночного паттерна (все конфигурации в одном месте), который со временем расширяется.

Возможно, вы заметили, что в главах 3 и 4 я сразу начала с паттернов модульной организации ресурсов инфраструктуры и зависимостей. Почему я не поставила эту главу о рефакторинге раньше? Если вы сможете выявить и применить некоторые паттерны на ранних стадиях разработки IaC, то сможете избежать монолитной архитектуры. Однако иногда вы получаете монолит в наследство, и вам приходится подвергать его рефакторингу.

10.2.1 Рефакторинг ресурсов высокого уровня

Команда Cap Sundew управляет сотнями ресурсов в одном наборе файлов конфигурации. С чего следует начать дробление IaC? Вы решаете искать *ресурсы высокого уровня* (high-level infra-structure resource), которые не зависят от других ресурсов.

У команды есть один набор высокоуровневой инфраструктуры в виде учетных записей и ролей службы IAM на уровне проекта GCP. Учетным записям и ролям служб IAM не требуется создавать сеть или сервер перед настройкой правил учетных записей пользователей и служб в проекте. Ни один из других ресурсов не зависит от ролей IAM и учетных записей служб. Вы можете сгруппировать их и выделить из общей инфраструктуры.

Вы не сможете применить blue-green подход к развертыванию, поскольку GCP не допускает дублирования политик. Но при этом вы не можете также скопировать их, удалить в монолитной конфигурации и вставить в новый репозиторий. Удаление учетных политик лишает всех возможности войти в проект! Как же их отделить?

Можно скопировать и вставить конфигурацию в отдельный репозиторий или каталог, инициализировать состояние отдельной конфигурации и импортировать ресурсы в состояние инфраструктуры, связанное с новой конфигурацией. После этого можно будет удалить конфигурацию IAM в монолитной инфраструктуре. Как и в случае с последовательным обновлением, вы постепенно меняете каждый набор ресурсов инфраструктуры, тестируете изменения и переходите к следующему набору.

На рис. 10.5 показан способ рефакторинга монолита для ресурсов высокого уровня. Вы копируете код из монолита в новый каталог и импортируете ресурс действующей инфраструктуры в состояние кода в этом каталоге. Вы повторно инициализируете (deploy) код, чтобы убедиться, что он не меняет существующую инфраструктуру. После этого можно удалить ресурсы высокого уровня из монолита.

Как и в случае с флагами функций, мы используем принцип идемпотентности для запуска IaC и проверки того, что мы не влияем на активное состояние инфраструктуры. Каждый раз, когда вы проводите рефакторинг, обязательно развертывайте изменения и делайте пробный прогон. Иначе вы рискуете случайно изменить существующий ресурс и повлиять на его зависимости.

В следующих нескольких разделах мы проведем рефакторинг примера. Не пропускайте эти разделы! Я знаю, что рефакторинг обычно кажется утомительным, но только последовательное изучение этой темы убережет вас от разрушительных последствий сбоя в системе.

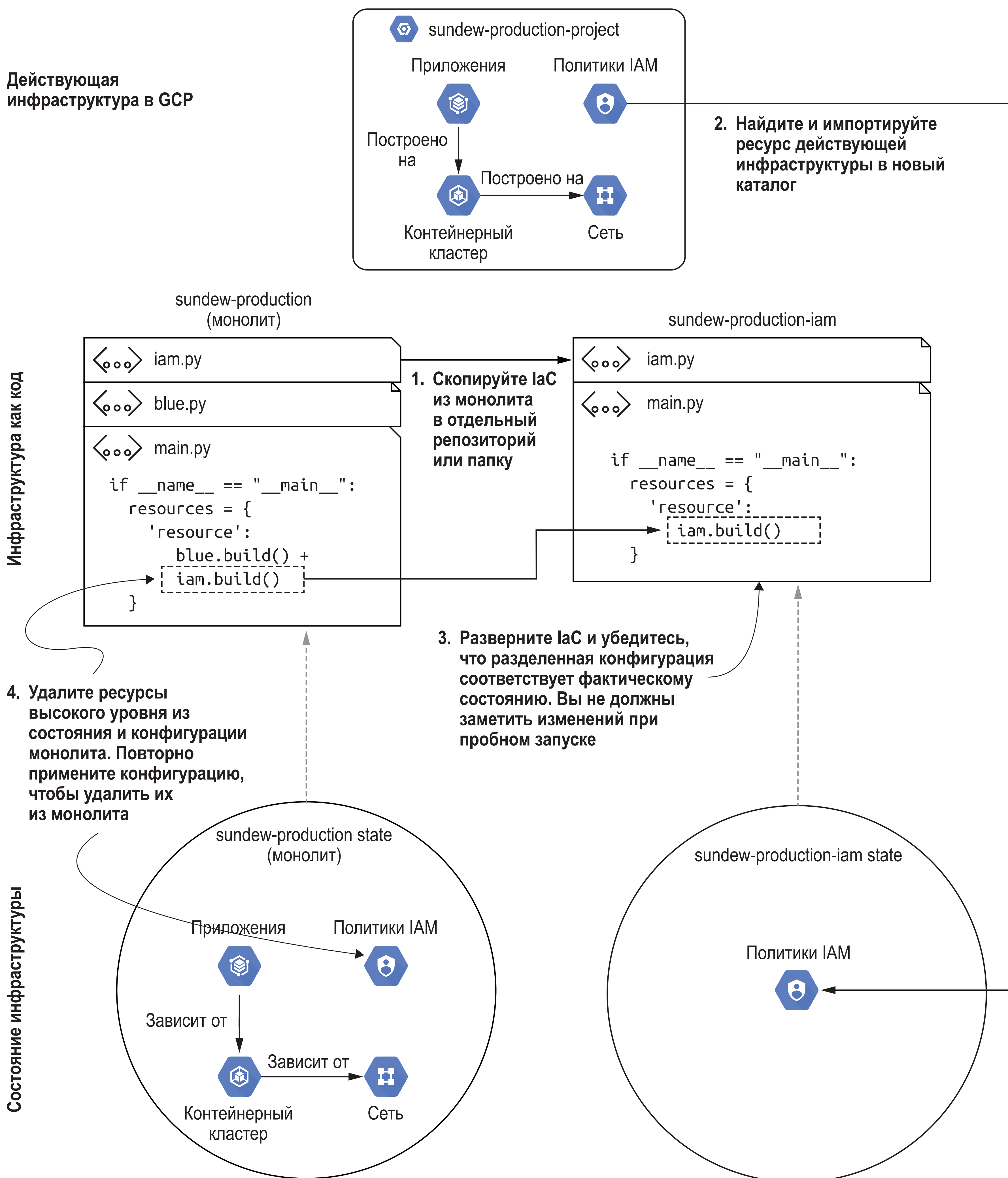


Рис. 10.5 IAM-политики системы Cap Sundew для проекта GCP не имеют зависимостей, и вы можете легко их реорганизовать, не нарушая работу другой инфраструктуры

Копирование из монолита в отдельное состояние

Рефакторинг начинается с копирования кода для создания ролей IAM и учетных записей служб в новый каталог. Команда Cap Sundew хочет сохранить структуру единого репозитория, в которой все IaC будут храниться в одном репозитории системы контроля версий, но конфигурации будут разделены на каталоги.

Выберите роли IAM и учетные записи служб, чтобы скопировать код команды в новый каталог, как показано на рис. 10.6. Активные политики IAM и состояние их инфраструктуры в GCP не изменяются.

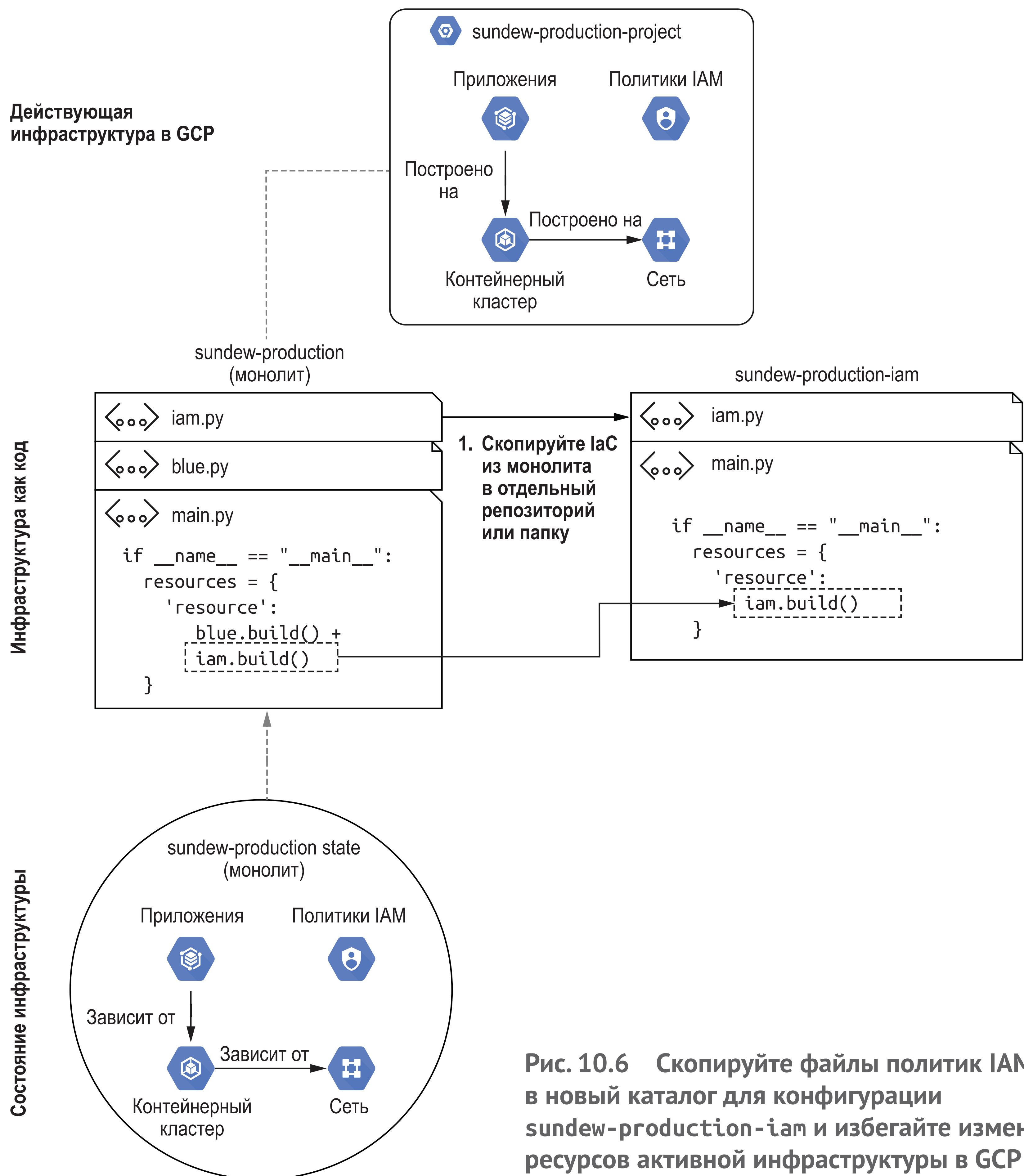


Рис. 10.6 Скопируйте файлы политик IAM в новый каталог для конфигурации sundew-production-iam и избегайте изменения ресурсов активной инфраструктуры в GCP

Зачем воспроизводить IaC для политик IAM в отдельном каталоге? Вам нужно разделить монолит IaC, не затрагивая ни один из активных ресурсов. Самая важная практика рефакторинга заключается в сохранении идемпотентности. Состояние действующей инфраструктуры никогда не должно меняться при перемещении IaC.

Приступим к рефакторингу политик IAM из монолита. Создайте новый каталог, который управляет только политиками IAM для проекта GCP:


```
$ mkdir -p sundew_production_iam
```

Скопируйте конфигурацию IAM из монолита в новый каталог:

```
$ cp iam.py sundew_production_iam/
```

Вам не нужно ничего менять, поскольку политики IAM не зависят от другой инфраструктуры. Файл `iam.py` в листинге 10.5 разделяет создание и назначение ролей для набора пользователей.

Листинг 10.5 Конфигурация IAM, отделенная от монолита

```
import json

TEAM = 'sundew'
TERRAFORM_GCP_SERVICE_ACCOUNT_TYPE = 'google_service_account' ①
TERRAFORM_GCP_ROLE_ASSIGNMENT_TYPE = 'google_project_iam_member' ①

users = { ②
    'audit-team': 'roles/viewer', ②
    'automation-watering': 'roles/editor', ②
    'user-02': 'roles/owner' ②
}

def get_user_id(user):
    return user.replace('-', '_')

def build():
    return iam()

def iam(users=users): ③
    iam_members = []
    for user, role in users.items():
        user_id = get_user_id(user)
        iam_members.append({
            TERRAFORM_GCP_SERVICE_ACCOUNT_TYPE: [{ ④
                user_id: [{ ④
                    'account_id': user, ④
                    'display_name': user ④
                }] ④
            }] ④
        })
        iam_members.append({
            TERRAFORM_GCP_ROLE_ASSIGNMENT_TYPE: [{ ⑤
                user_id: [{ ⑤
                    'role': role, ⑤
                    'member': 'serviceAccount:${google_service_account.' ⑤
                    + f'{user_id}' + '.email}' ⑤
                }] ⑤
            }] ⑤
        })
    return iam_members
```


- ❶ Устанавливаем типы ресурсов, которые Terraform использует в качестве констант, чтобы при необходимости вы могли ссылаться на них позже.
- ❷ Сохраняем всех пользователей, которых вы добавили в проект, как часть монолита.
- ❸ Применяем модуль для создания конфигурации JSON политик IAM вне монолита.
- ❹ Создаем учетную запись службы GCP для каждого пользователя в проекте по выращиванию росянки.
- ❺ Назначаем конкретную роль, определенную для каждой учетной записи службы, например просмотр, изменение или владелец.

Эквиваленты AWS и Azure

В листинге 10.5 все пользователи и группы создаются как учетные записи служб в GCP, поэтому вы можете выполнить пример до конца. Обычно учетные записи служб используют для автоматизации.

Учетная запись службы в GCP аналогична пользователю AWS IAM, предназначенному для автоматизации служб, или приложению Azure Active Directory, зарегистрированному с секретным ключом клиента. Чтобы перенести код в AWS или Azure, настройте роли viewer (только просмотр), editor (редактирование) и owner (владелец) в соответствии с ролями AWS или Azure.

Определите константы и создайте методы, которые выводят типы и идентификаторы ресурсов при разделении конфигурации. Вы всегда можете использовать их для другой автоматизации и дальнейшего обслуживания системы, особенно если вы продолжаете рефакторинг монолита!

Создайте файл main.py (листинг 10.6) в каталоге sundew_production_iam, который обращается к конфигурации IAM и формирует для нее файл Terraform в формате JSON.

Листинг 10.6 Точка входа для создания отдельной конфигурации JSON для IAM

```
import iam ❶
import json

if __name__ == "__main__":
    resources = {
        'resource': iam.build() ❶
    }

    with open('main.tf.json', 'w') as outfile: ❷
        json.dump(resources, outfile, ❷
                    sort_keys=True, indent=4) ❷
```

- ❶ Импортируем код конфигурации IAM и создаем политики IAM.
- ❷ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Пока не запускайте код Python для создания Terraform JSON или развертывания политик IAM! У вас уже есть политики IAM, определенные как часть GCP. Если вы выполните команду `python main.py` и примените Terraform JSON с отдельной конфигурацией IAM, GCP выдаст ошибку о том, что учетная запись пользователя и привязка уже существуют:

```
$ python main.py
$ terraform apply -auto-approve
## output omitted for clarity
| Error: Error creating service account: googleapi:
➔Error 409: Service account audit-team already exists within project
➔projects/infrastructure-as-code-book., alreadyExists
```

Членам команды Car Sundew не нужно, чтобы вы удаляли и создавали новые учетные записи и роли. Если вы удалите и заново создадите учетные записи, они не смогут войти в свой проект GCP.

Вам нужен способ перенести существующие ресурсы, определенные в монолите, и связать их с кодом, определенным в отдельном каталоге.

Импорт ресурсов в новое состояние

Иногда создание новых ресурсов с помощью реорганизованной IaC способно нарушить работу команд и критически важных для бизнеса систем. В таком случае вы не можете использовать принцип неизменяемости для удаления старых ресурсов и создания новых. Вместо этого вам необходимо перенести активные ресурсы из одного определения IaC в другое.

В примере с командой Car Sundew вы извлекаете идентификаторы для каждой сервисной учетной записи из монолитной конфигурации и «перемещаете» их в новое состояние. На рис. 10.7 показано, как отсоединить каждую учетную запись службы и ее назначения ролей от монолита и присоединить их к IaC в каталоге `sundew_production_iam`. Вы вызываете API GCP для получения текущего состояния политик IAM и импортируете действующие ресурсы инфраструктуры в отдельную конфигурацию и состояние. Запуск IaC не должен выявить никаких изменений в пробном прогоне.

Зачем импортировать информацию о политике IAM с помощью GCP API? Вам необходимо импортировать обновленное активное состояние ресурса. API облачного провайдера предлагает самую актуальную конфигурацию ресурсов. Вы можете вызвать API GCP, чтобы получить адреса электронной почты, роли и идентификаторы пользователей для команды Car Sundew.

Вместо того чтобы разрабатывать собственный инструмент импорта и сохранять идентификаторы в файле, вы решаете использо-

вать возможность импорта Terraform для добавления существующих ресурсов в состояние. В листинге 10.7 представлен код Python, который является оберткой функций Terraform и автоматизирует пакетный импорт ресурсов IAM, чтобы команда Cap Sundew могла повторно использовать его.

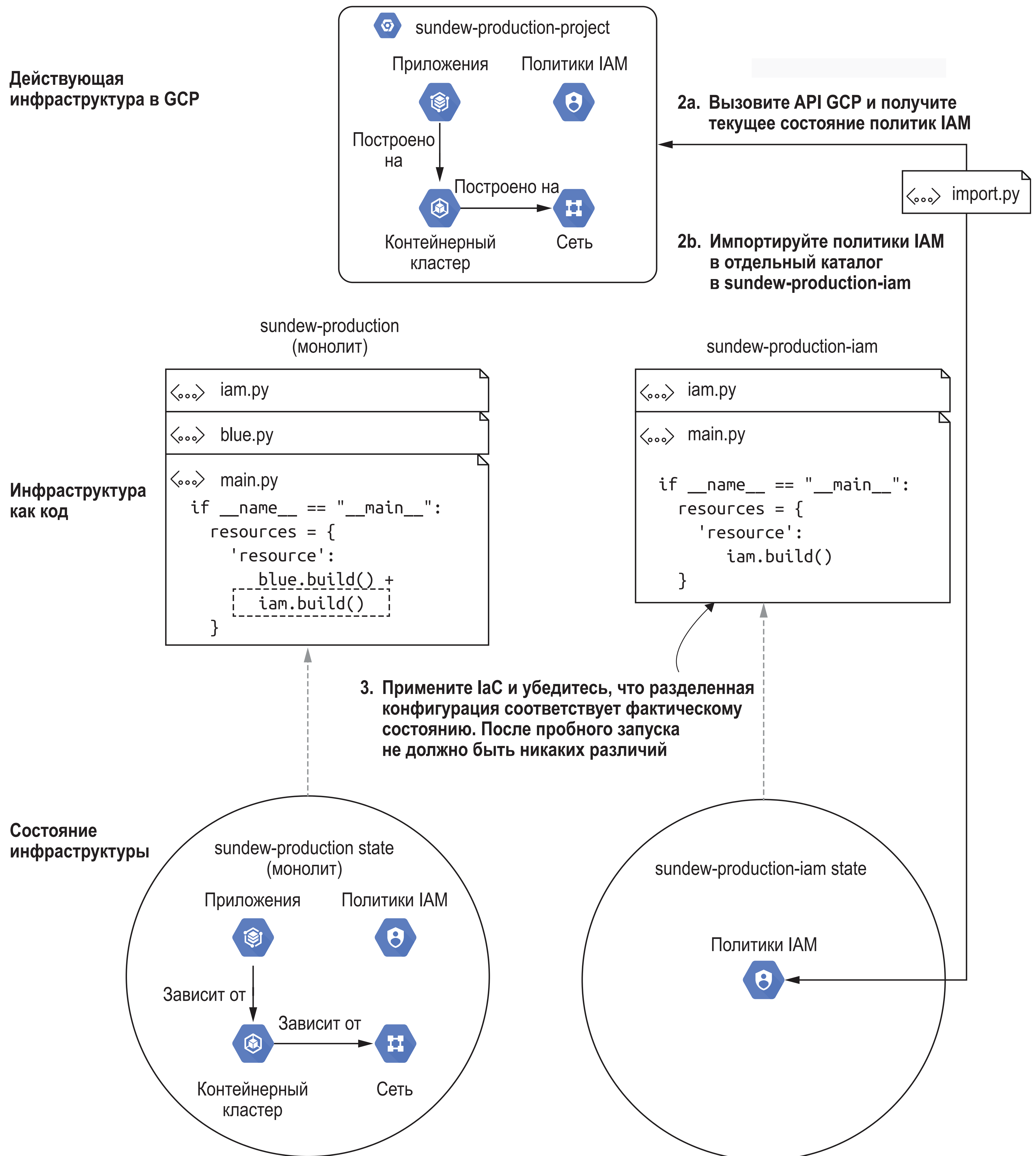


Рис. 10.7 Получение текущего состояния отдельных ресурсов от провайдера инфраструктуры и импорт идентификаторов перед повторным применением IaC

Листинг 10.7 Файл `import.py` отдельно импортирует ресурсы IAM росянки

```

import iam 1
import os
import googleapiclient.discovery 2
import subprocess

PROJECT = os.environ['CLOUDSDK_CORE_PROJECT'] 3

def _get_members_from_gcp(project, roles): 2
    roles_and_members = {} 2
    service = googleapiclient.discovery.build( 2
        'cloudresourcemanager', 'v1') 2
    result = service.projects().getIamPolicy( 2
        resource=project, body={}).execute() 2
    bindings = result['bindings'] 2
    for binding in bindings: 2
        if binding['role'] in roles: 2
            roles_and_members[binding['role']] = binding['members'] 2
    return roles_and_members 2

def _set_emails_and_roles(users, all_members): 4
    members = [] 4
    for username, role in users.items(): 4
        members += [(iam.get_user_id(username), m, role) 4
                     for m in all_members[role] if username in m] 4
    return members 4

def check_import_status(ret, err):
    return ret != 0 and \
        'Resource already managed by Terraform' not in str(err)

def import_service_account(project_id, user_id, user_email): 5
    email = user_email.replace('serviceAccount:', '') 5
    command = ['terraform', 'import', '-no-color', 5
               f'{iam.TERRAFORM_GCP_SERVICE_ACCOUNT_TYPE}.{user_id}', 5
               f'projects/{project_id}/serviceAccounts/{email}'] 5
    return _terraform(command) 5

def import_project_iam_member(project_id, role, user_id, user_email): 6
    command = ['terraform', 'import', '-no-color', 6
               f'{iam.TERRAFORM_GCP_ROLE_ASSIGNMENT_TYPE}.{user_id}', 6
               f'{project_id} {role} {user_email}'] 6
    return _terraform(command) 6

def _terraform(command): 7
    process = subprocess.Popen( 7
        command, 7
        stdout=subprocess.PIPE, 7
        stderr=subprocess.PIPE) 7
    stdout, stderr = process.communicate() 7
    return process.returncode, stdout, stderr 7

if __name__ == "__main__":

```



```

sundew_iam = iam.users
all_members_for_roles = _get_members_from_gcp(
    PROJECT, set(sundew_iam.values()))
import_members = _set_emails_and_roles(
    sundew_iam, all_members_for_roles)
for user_id, email, role in import_members:
    ret, _, err = import_service_account(PROJECT, user_id, email)
    if check_import_status(ret, err):
        print(f'import service account failed: {err}')
    ret, _, err = import_project_iam_member(PROJECT, role, user_id, email)
    if check_import_status(ret, err):
        print(f'import iam member failed: {err}')

```

- ❶ Получаем список пользователей команды Cap Sundew из iam.py в sundew_production_iam.
- ❷ Используем клиентские библиотеки Google Cloud для Python для получения списка участников, которым назначена роль в проекте GCP.
- ❸ Получаем идентификатор проекта GCP из переменной среды CLOUDSDK_CORE_PROJECT.
- ❹ Получаем адрес электронной почты и идентификаторы пользователей только для членов Sundew IAM.
- ❺ Импортируем учетную запись службы в состояние sundew_production_iam на основе адреса электронной почты проекта и пользователя, используя константу типа ресурса, заданную вами в iam.py.
- ❻ Импортируем назначение роли в состояние sundew_production_iam на основе проекта, роли и адреса электронной почты пользователя, используя константу типа ресурса, заданную вами в iam.py.
- ❼ Оба метода импорта обертывают команду Terraform CLI и возвращают любые ошибки и выходные данные.
- ❽ Получаем список пользователей команды Sundew из iam.py в sundew_production_iam.
- ❾ Используем клиентские библиотеки Google Cloud для Python для получения списка участников, которым назначена роль в проекте GCP.
- ❿ Получаем адрес электронной почты и идентификаторы пользователей только для членов Sundew IAM.
- ⓫ Если импорт не удался и ресурс еще не импортирован, выводится сообщение об ошибке.

Libcloud или SDK облачных провайдеров

В примерах в этой главе необходимо использовать клиентскую библиотеку Google Cloud для Python вместо Apache Libcloud, который я показал в главе 4. Хотя библиотека Apache Libcloud позволяет получать информацию о виртуальных машинах, она не подходит для других ресурсов в GCP. Дополнительную информацию о клиентской библиотеке Google Cloud для Python можно найти по адресу <http://mng.bz/1ooZ>.

Обновите листинг 10.7, чтобы использовать библиотеки Azure для Python (<http://mng.bz/Pnn2>) или AWS SDK для Python (<https://aws.amazon.com/sdk-for-python/>) для получения информации о пользователях. Они заменят клиентскую библиотеку GCP API.

Как и при определении зависимостей, необходимо *динамически получать* идентификаторы из API поставщика инфраструктуры для

импорта ваших ресурсов. Невозможно знать наперед, когда кто-то изменит ресурс, а идентификатор, который вы рассчитывали получить, перестанет существовать! Используйте свои теги и политику именования объектов для поиска в ответе API необходимых вам ресурсов.

Когда вы запускаете скрипт командой `python import.py` и выполняете пробный прогон Terraform JSON с отдельной конфигурацией IAM, вы получаете сообщение о том, что вам не нужно вносить какие-либо изменения. Вы успешно импортировали существующие ресурсы IAM в их отдельную конфигурацию и состояние:

```
$ python main.py
```

```
$ terraform plan
```

```
No changes. Your infrastructure matches the configuration.
```

```
Terraform has compared your real infrastructure against your configuration  
➡and found no differences, so no changes are needed.
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Иногда пробный прогон указывает на расхождение между состоянием активного ресурса и измененной/новой конфигурацией. Это означает, что скопированная конфигурация не соответствует состоянию активного ресурса. Различия часто возникают из-за того, что кто-то изменил состояние ресурса инфраструктуры вручную, в то время пока вы выполняли копирование и перенос. Обновите выделенную конфигурацию IaC, чтобы она *соответствовала атрибутам активного ресурса инфраструктуры*.

Импорт с помощью инструмента конфигурирования и без него

Многие инструменты конфигурирования имеют функцию импорта ресурсов. Например, AWS CloudFormation использует команду `resource import`. В примере для перемещения учетных записей служб используется код Python, обернутый вокруг `terraform import`. Без него дробить монолитную конфигурацию было бы утомительно.

Если вы разрабатываете IaC без инструмента, вам не нужна возможность прямого импорта. Вместо этого вам нужна логика для проверки существования ресурсов. Учетные записи служб выращивания росы и назначения ролей из нашего примера могут работать без применения импорта Terraform или IaC:

- 1 вызовите GCP API, чтобы проверить, существуют ли сервисные учетные записи и вложения ролей команды Cap Sundew;
- 2 если да, проверьте, соответствует ли ответ API для атрибутов учетной записи службы желаемой конфигурации. При необходимости обновите учетную запись службы;
- 3 если нет, создайте учетные записи служб и вложения ролей.

Удаление выделенных ресурсов из монолита

Вам удалось извлечь и переместить учетные записи и назначения ролей команды Cap Sundew в отдельную конфигурацию IaC. Но вы не хотите, чтобы ресурсы оставались в монолите. Вы удаляете ресурсы из *состояния и конфигурации* монолита перед повторным применением и обновлением инструмента, как показано на рис. 10.8.

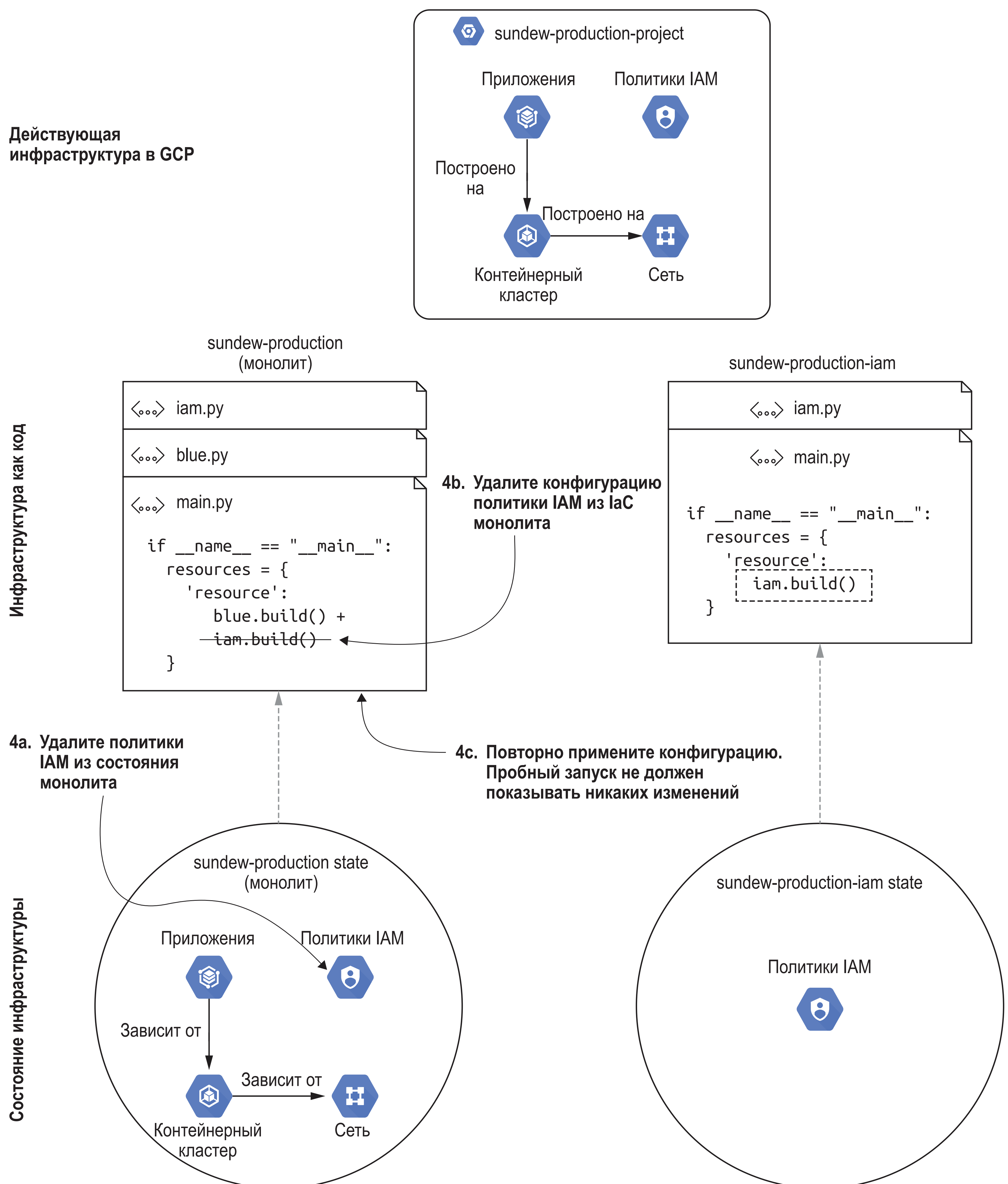


Рис. 10.8 Удаление политик из состояния и конфигурации монолита перед применением обновлений и завершением рефакторинга

Этот шаг помогает соблюдать гигиену IaC. В главе 2 было сказано, что ваша IaC должна служить своего рода *источником истинного состояния*. Вы не должны управлять одним ресурсом с помощью двух IaC. Если они конфликтуют, два определения IaC для ресурса могут повлиять на зависимости и конфигурацию системы.

Вам нужно, чтобы источником истинного состояния служил каталог политики IAM. В дальнейшем команде Cap Sundew необходимо будет вносить обновления в политику IAM, выделенную в отдельный каталог, а не в монолите. Чтобы избежать путаницы, нужно удалить ресурсы IAM из монолита IaC.

Для начала вам необходимо удалить ресурсы IAM команды Cap Sundew из состояния Terraform, представленного в файле JSON. Terraform имеет команду удаления состояния, которую вы можете использовать для извлечения частей JSON на основе идентификатора ресурса. В листинге 10.8 код Python служит оболочкой команды Terraform. Код позволяет вам передавать любой тип ресурса и идентификатор, который вы хотите удалить из состояния инфраструктуры.

Листинг 10.8 Файл remove.py удаляет ресурсы из состояния монолита

```

from sundew_production_iam import iam                                1
import subprocess

def check_state_remove_status(ret, err):                             2
    return ret != 0 \                                              2
        and 'No matching objects found' not in str(err)           2

def state_remove(resource_type, resource_identifier):               3
    command = ['terraform', 'state', 'rm', '-no-color',            3
               f'{resource_type}.{resource_identifier}']          3
    return _terraform(command)                                     3

def _terraform(command):                                            4
    process = subprocess.Popen(
        command,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
    stdout, stderr = process.communicate()
    return process.returncode, stdout, stderr

if __name__ == "__main__":
    sundew_iam = iam.users                                         1
    for user in iam.users:                                         5
        ret, _, err = state_remove(                               6
            iam.TERRAFORM_GCP_SERVICE_ACCOUNT_TYPE,              6
            iam.get_user_id(user))                                6
        if check_state_remove_status(ret, err):                   7
            print(f'remove service account from state failed: {err}') 7
        ret, _, err = state_remove(                               8
            iam.TERRAFORM_GCP_ROLE_ASSIGNMENT_TYPE,              8

```



```

iam.get_user_id(user))
if check_stat_remove_status(ret, err):
    print(f'remove role assignment from state failed: {err}')

```

- ❶ Извлекаем список пользователей команды Cap Sundew из iam.py в sundew_production_iam. Ссылка на переменную из отдельного IaC позволяет запустить автоматизацию удаления для будущих действий по рефакторингу.
- ❷ Если удаление не удалось и ресурс еще не был удален, выводится сообщение об ошибке.
- ❸ Создаем метод, который оборачивает команду удаления состояния Terraform. Команда передает тип ресурса, например учетную запись службы и идентификатор, который необходимо удалить.
- ❹ Открываем подпроцесс, который запускает команду Terraform для удаления ресурса из состояния.
- ❺ Для каждого пользователя в sundew_production_iam удаляем его учетную запись службы и назначение ролей из состояния монолита.
- ❻ Удаляем учетную запись службы GCP из состояния Terraform монолита на основе ее идентификатора пользователя.
- ❼ Проверяем, что команда Terraform успешно удалила ресурс из состояния монолита.
- ❽ Удаляем назначение роли GCP из состояния Terraform монолита на основе его идентификатора пользователя.

Пока не выполняйте команду `python remove.py`! Ваш монолит по-прежнему содержит определение политик IAM. Откройте файл `main.py` вашего монолитного IaC. Как показано в листинге 10.9, удалите код, который создает учетные записи службы IAM и назначения ролей для членов команды Cap Sundew.

Листинг 10.9 Удаление политик IAM из кода монолита

```

import blue
import json

if __name__ == "__main__":
    resources = {
        'resource': blue.build()

    with open('main.tf.json', 'w') as outfile:
        json.dump(resources, outfile,
                   sort_keys=True, indent=4)

```

- ❶ Удаляем импорт политик IAM.
- ❷ Удаляем код для создания политик IAM внутри монолита, оставляя другие ресурсы.
- ❸ Записываем конфигурацию в файл JSON, который будет выполнен Terraform позже. Конфигурация не включает политики IAM.

Теперь вы можете обновить свой монолит. Сначала выполните `python remove.py`, чтобы удалить ресурсы IAM из состояния монолита:

```
$ python remove.py
```


С этого момента ваш монолит больше не служит источником информации для политик IAM и учетных записей служб. Вы *не удаляете* ресурсы IAM! Вы можете рассматривать это как передачу права собственности на ресурсы IAM новому IaC в отдельном каталоге.

Теперь вы наконец можете обновить монолит в своем терминале. Создайте новый файл Terraform JSON без политик IAM и примените обновления; у вас не должно быть никаких изменений:

```
$ python main.py
```

```
$ terraform apply
```

```
google_service_account.blue: Refreshing state...
google_compute_network.blue: Refreshing state...
google_compute_subnetwork.blue: Refreshing state...
google_container_cluster.blue: Refreshing state...
google_container_node_pool.blue: Refreshing state...
```

```
No changes. Your infrastructure matches the configuration.
```

```
Terraform has compared your real infrastructure against your configuration
➡and found no differences, so no changes are needed.
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Если пробный прогон показывает наличие ресурсов, подвергнутых рефакторингу, это означает, что вы не удалили их из состояния или конфигурации монолита. Вам необходимо изучить ресурсы и определить, следует ли удалять их вручную.

10.2.2 Рефакторинг ресурсов с зависимостями

Теперь вы можете перейти к ресурсам инфраструктуры нижнего уровня с зависимостями, такими как контейнерный оркестратор команды Cap Sundew. Члены команды просят вас не создавать новый оркестратор и не уничтожать старый, поскольку они не хотят нарушать работу приложений. Вам необходимо выполнить рефакторинг и извлечь низкоуровневый оркестратор буквально «на ходу».

Начните копировать конфигурацию контейнера из монолита, повторяя тот же процесс, который вы использовали для рефакторинга учетных записей и ролей службы IAM. Создайте отдельный каталог с названием `sundew_production_orchestrator`:

```
$ mkdir -p sundew_production_orchestrator
```

Вы выбираете и копируете метод создания кластера в `sundew_production_orchestrator/cluster.py`. Однако у вас есть проблема. Оркестратору контейнера нужны имена сети и подсети. Как получить имя сети и подсети, если оркестратор контейнера не может обращаться к монолиту?

На рис. 10.9 показана реализация внедрения зависимостей в существующий монолит с использованием API провайдера инфраструктуры в качестве уровня абстракции. Для создания кластера IaC вызывает API GCP и получает информацию о сети. Вы передаете идентификатор сети кластеру для дальнейшего использования.

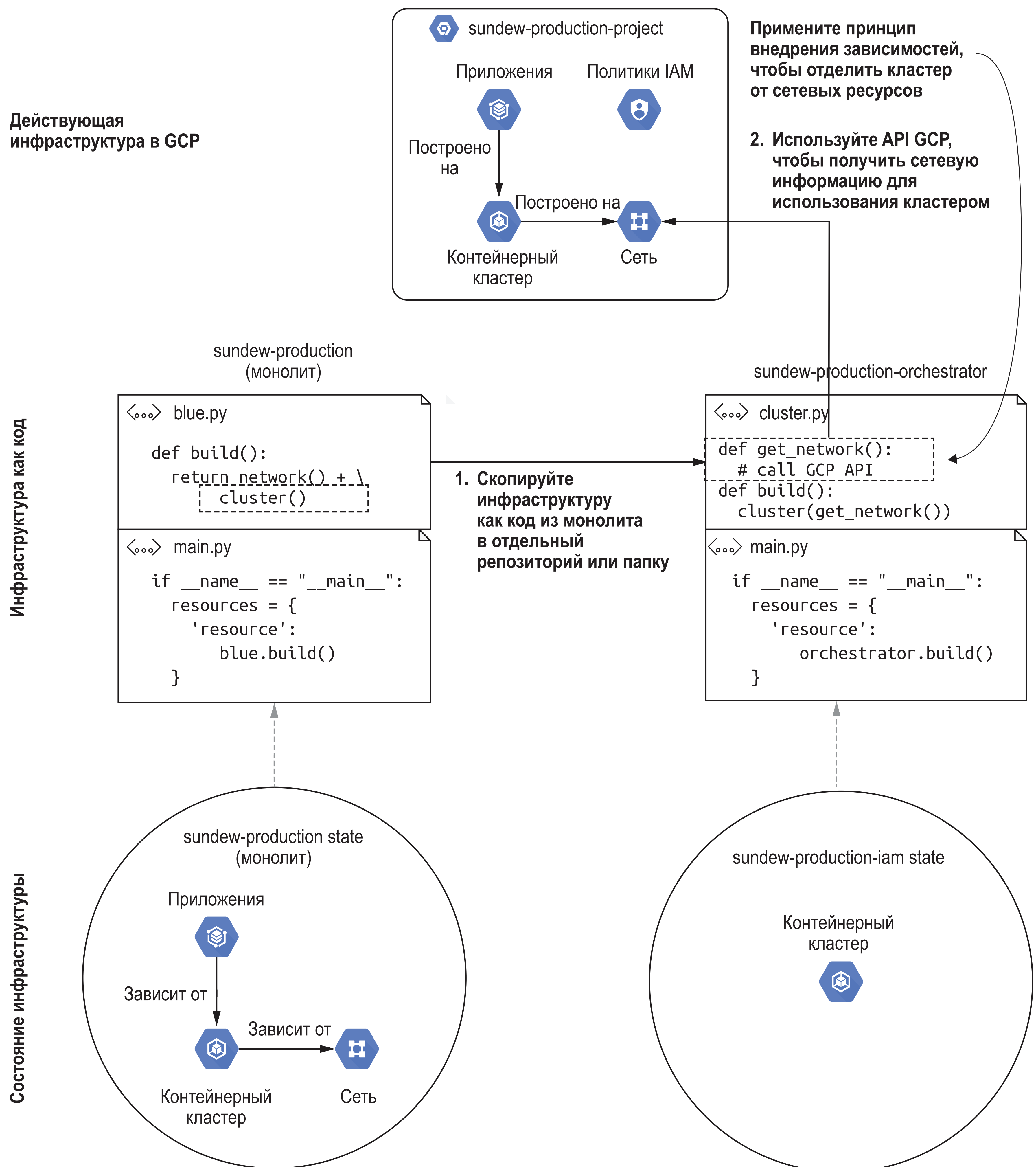


Рис. 10.9 Скопируйте инфраструктуру и добавьте новые методы для вызова API GCP и получения идентификатора сети для кластера

Монолит в явном виде отражает зависимость между ресурсами. Когда вы создаете новый каталог, вашим разделенным ресурсам

нужна информация о низкоуровневых зависимостях. Напомним, что вы можете разделить инфраструктурные модули с помощью внедрения зависимостей (см. главу 4). Модуль высокого уровня вызывает уровень абстракции, чтобы получить идентификаторы для зависимостей низкого уровня.

В ходе рефакторинга ресурсов с зависимостями вы должны реализовать интерфейс для внедрения зависимостей. В листинге 10.10 показано обновление кода из файла `sundew_production_orchestrator/cluster.py` для использования клиентской библиотеки Google Cloud и получения имен подсети и сети для конфигурации кластера.

ПРИМЕЧАНИЕ Некоторые зависимости, переменные и импорты были удалены из листинга 10.10 для большей ясности. Полный пример можно найти в репозитории кода книги по адресу <https://github.com/joatmon08/manning-book/tree/main/ch10/s03/s02>.

Листинг 10.10 Использование инверсии зависимостей для имени сети в кластере

```
import googleapiclient.discovery 1
def _get_network_from_gcp(): 2
    service = googleapiclient.discovery.build( 1
        'compute', 'v1') 1
    result = service.subnetworks().list( 3
        project=PROJECT, 3
        region=REGION, 3
        filter=f'name:"{TEAM}-{ENVIRONMENT}-*"').execute() 3
    subnetworks = result['items'] if 'items' in result else None

    if len(subnetworks) != 1: 4
        print("Network not found") 4
        exit(1) 4
    return subnetworks[0]['network'].split('/')[0], \ 5
        subnetworks[0]['name'] 5

def cluster(name=cluster_name, 6
            node_name=cluster_nodes,
            service_account=cluster_service_account,
            region=REGION):
    network, subnet = _get_network_from_gcp() 7
    return [
        {
            'google_container_cluster': { 8
                'VERSION': [
                    {
                        'name': name,
                        'network': network, 9
```



```

        'subnetwork': subnet
    },
    ],
},
'google_container_node_pool': {
    VERSION: [
        {
            'cluster':
                '${google_container_cluster.' +
                f'{VERSION}' + '.name}',
        }
    ]
},
'google_service_account': {
    VERSION: [
        {
            'account_id': service_account,
            'display_name': service_account
        }
    ]
}
}
]

```

- ❶ Настраиваем доступ к API GCP с помощью клиентской библиотеки Google Cloud для Python.
- ❷ Создаем метод, который извлекает сетевую информацию из GCP и реализует внедрение зависимостей.
- ❸ Запрашиваем у API GCP список подсетей, имена которых начинаются с `sun-dew-production`.
- ❹ Выдаем ошибку, если API GCP не нашел подсеть.
- ❺ Возвращаем имя сети и имя подсети.
- ❻ Некоторые зависимости, переменные и импорты были удалены из листинга кода для большей ясности. Полный пример можно найти в репозитории кода книги.
- ❼ Применяем принцип инверсии зависимостей и вызываем API GCP для получения имен сети и подсети.
- ❽ Создаем кластер контейнеров Google, пул узлов и учетную запись службы с помощью ресурса Terraform.
- ❾ Используем имена сети и подсети для обновления кластера контейнеров.

Эквиваленты AWS и Azure

Вы можете обновить листинг 10.10, чтобы использовать библиотеки Azure для Python (<http://mng.bz/Pnn2>) или AWS SDK для Python (<https://aws.amazon.com/sdk-for-python/>) для замены клиентской библиотеки API GCP.

Далее обновите ресурсы. Создайте Amazon VPC и виртуальную сеть Azure для пулов узлов Kubernetes (также называемых *группами*). Затем переключите кластер контейнеров Google на кластер Amazon EKS или AKS.

При рефакторинге ресурса инфраструктуры с зависимостями необходимо реализовать внедрение зависимостей для получения атрибутов ресурса нижнего уровня. В листинге 10.10 задействован API провайдера инфраструктуры, но вы можете использовать любой уровень абстракции по вашему выбору. API провайдера инфраструктуры часто обеспечивает наиболее простую абстракцию. Вы можете использовать его, чтобы не разрабатывать собственную реализацию.

После копирования и обновления кластера контейнеров для ссылки на имена сетей и подсетей из API GCP вы повторяете рабочий процесс рефакторинга, показанный на рис. 10.10. Вы импортируете ресурс активной инфраструктуры в `sundew_production_orchestrator`, применяете отдельную конфигурацию, проверяете наличие расхождений между активным состоянием и IaC и удаляете конфигурацию и ссылку ресурса в состоянии монолита.

Основное различие между рефакторингом ресурсов высокого уровня и ресурсов более низкого уровня из монолита заключается в реализации внедрения зависимостей. Вы можете выбрать тип внедрения зависимостей, который хотите использовать, например API поставщика инфраструктуры, выходные данные модуля или состояние инфраструктуры. Имейте в виду, что вам может потребоваться изменить монолитный IaC для вывода атрибутов, если вы не используете API поставщика инфраструктуры.

В противном случае убедитесь в соблюдении принципа идемпотентности, перезапустив IaC после рефакторинга. Вы должны избежать влияния на активные ресурсы и изолировать все изменения в IaC. Если пробный прогон показывает наличие изменений, вам необходимо устранить разницу между кодом после рефакторинга и состоянием активной инфраструктуры, прежде чем переходить к другим ресурсам.

10.2.3 *Повторение процесса рефакторинга*

После извлечения учетных записей и ролей службы IAM, а также оркестратора контейнеров вы можете продолжить дробление монолитной конфигурации IaC системы. Рабочий процесс на рис. 10.11 обобщает схему дробления монолитного IaC. Вы определяете, какие ресурсы зависят друг от друга, извлекаете их конфигурацию и обновляете их зависимости для использования внедрения зависимостей.

Найдите ресурсы инфраструктуры высокого уровня, которые ни от чего не зависят и не имеют зависимых ресурсов. Я использую такие ресурсы для тестирования рабочего процесса копирования, разделения, импорта и удаления из монолита. Далее я нахожу ресурсы высокого уровня, которые зависят от других ресурсов. Во время копирования я реорганизирую их, чтобы ссылаться на атрибуты посред-

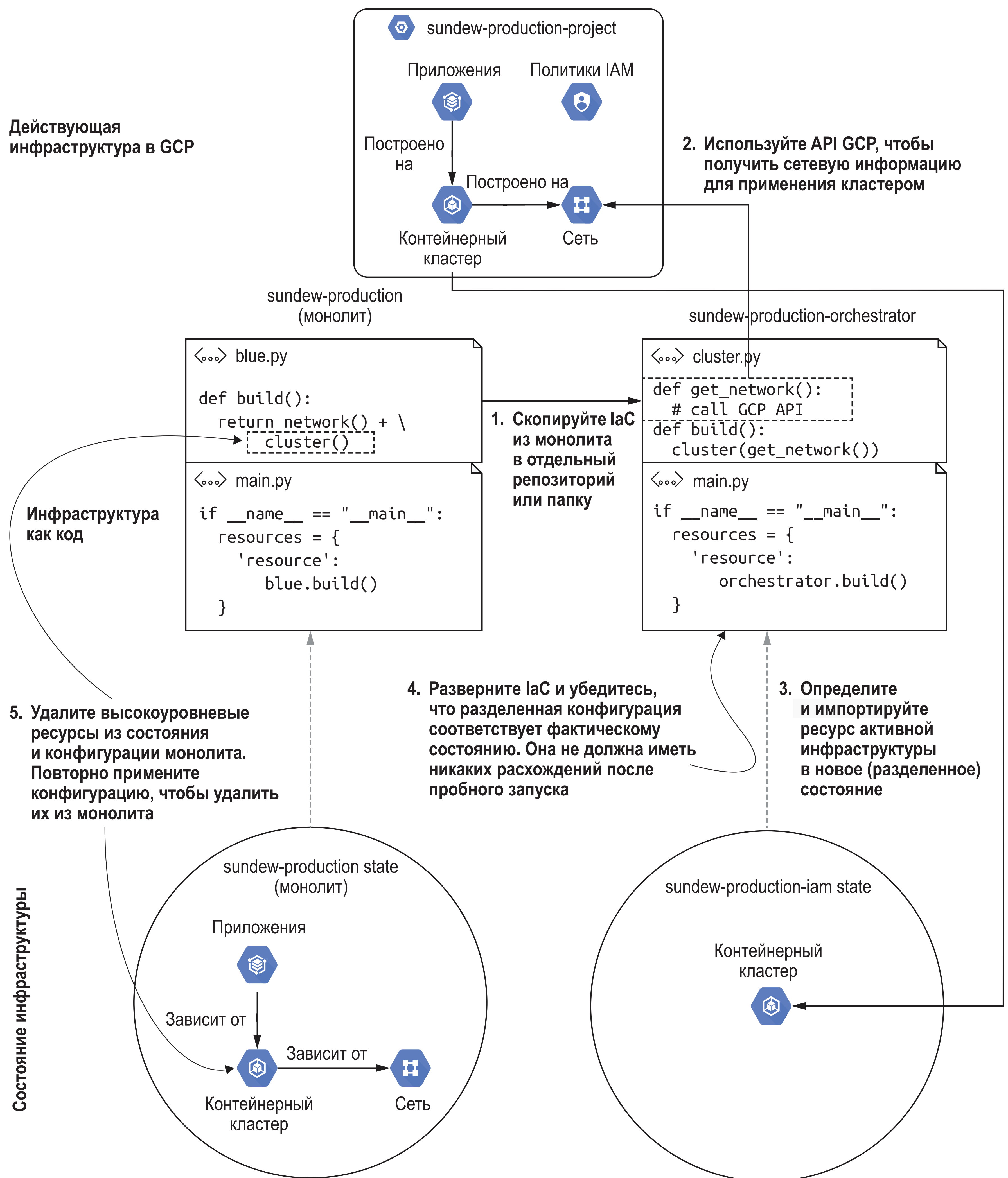


Рис. 10.10 Рефакторинг ресурсов более высокого уровня для получения идентификаторов низкого уровня с помощью GCP API, выполняемый перед рефакторингом ресурсов низкого уровня

ством внедрения зависимостей. Я итеративно повторяю процесс, в конечном итоге заканчивая ресурсами самого низкого уровня, не имеющими никаких зависимостей.

Как выполнить рефакторинг монолита IaC

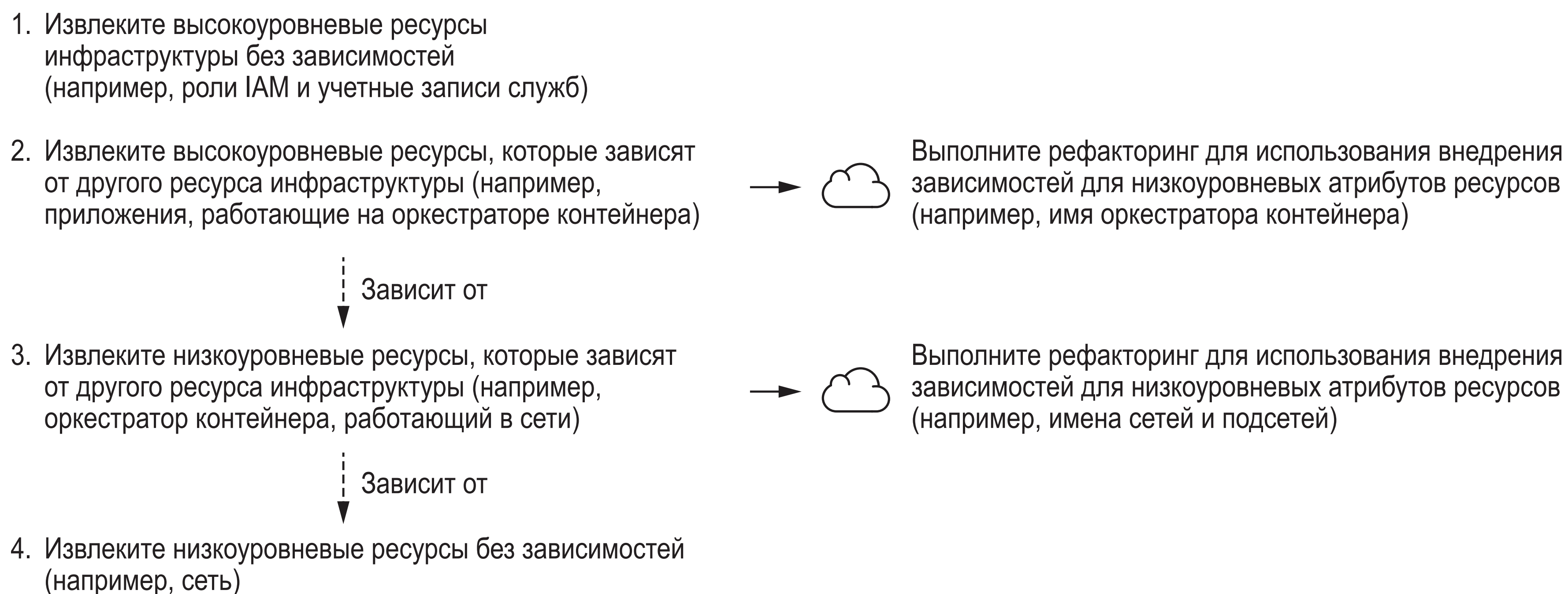


Рис. 10.11 Рабочий процесс рефакторинга монолита IaC начинается с определения ресурсов высокого уровня, не имеющих каких-либо зависимостей

Управление конфигурацией

Хотя в этой главе основное внимание уделяется инструментам конфигурирования IaC, управление конфигурацией также может превратиться в монолит и привести к тем же проблемам, включая длительное время выполнения или наличие конфликтующих изменений в разных частях одной конфигурации. Вы можете применить аналогичный рабочий процесс рефакторинга к монолитному управлению конфигурацией:

- 1 Извлеките наиболее независимые компоненты автоматизации без каких-либо зависимостей и отделите их в модуль.
- 2 Запустите диспетчер конфигурации и убедитесь, что вы не изменили состояние ресурса.
- 3 Найдите конфигурацию, которая зависит от выходных данных или наличия средств автоматизации нижнего уровня. Извлеките ее и примените внедрение зависимостей, чтобы получить все необходимые значения для конфигурации.
- 4 Запустите менеджер конфигурирования и убедитесь, что вы не изменили состояние ресурса.
- 5 Повторяйте этот процесс до тех пор, пока не дойдете до первого шага вашего менеджера конфигурирования.

При рефакторинге монолитов IaC тщательно продумайте способы отделения ресурсов друг от друга. Я считаю, что рефакторинг – это сложный процесс, и редко когда он обходится без сбоев и ошибок. Изолирование отдельных компонентов и их тщательное тестирование помогут выявить проблемы и свести к минимуму сбои в работе системы. Если я сталкиваюсь с неудачами, я использую методы, описанные в главе 11, чтобы исправить их.

Упражнение 10.1

Какой порядок и группировку ресурсов вы бы использовали для рефакторинга и дробления монолита после анализа следующего кода:

```
if __name__ == "__main__":
    zones = ['us-west1-a', 'us-west1-b', 'us-west1-c']
    project.build()
    network.build(project)
    for zone in zones:
        subnet.build(project, network, zone)
        database.build(project, network)
    for zone in zones:
        server.build(project, network, zone)
        load_balancer.build(project, network)
    dns.build()
```

- a** DNS, балансировщик нагрузки, серверы, база данных, сеть + подсети, проект;
- b** балансировщик нагрузки + DNS, база данных, серверы, сеть + подсети, проект;
- c** проект, сеть + подсети, серверы, база данных, балансировщик нагрузки + DNS;
- d** база данных, балансировщик нагрузки + DNS, серверы, сеть + подсети, проект.

Ответы на упражнения даны в приложении В.

Краткое содержание главы

- Рефакторинг IaC представляет собой реструктуризацию конфигурации или кода без воздействия на существующие ресурсы инфраструктуры.
- Рефакторинг устраняет технический долг – метафорическое название стоимости изменения кода с накопившимися проблемами.
- При использовании стратегии последовательного обновления одинаковые ресурсы инфраструктуры заменяются один за другим. Каждый обновленный ресурс тестируют, прежде чем перейти к следующему.
- Последовательное обновление позволяет поэтапно внедрять изменения и устранять неполадки.
- Флаги функций (feature flags, переключатели функций) включают или отключают ресурсы инфраструктуры, зависимости или атрибуты.
- Применяйте флаги функций для тестирования, подготовки и скрывания изменений перед их применением в рабочей среде.
- Соберите флаги функций в одном месте (например, в файле или менеджере конфигурации), чтобы сразу определить их значения.

- Удалите флаги функций, если они вам больше не нужны.
- Монолитная структура IaC возникает, когда вы определяете все ресурсы инфраструктуры в одном месте и удаление одного ресурса приводит к сбою всей конфигурации.
- Рефакторинг ресурса для извлечения из монолита состоит из разделения и копирования конфигурации в новый каталог или репозиторий, импорта ее в новое отдельное состояние и удаления ресурсов из монолитной конфигурации и состояния.
- Если ваш ресурс зависит от другого ресурса, обновите конфигурацию отдельного ресурса, чтобы использовать внедрение зависимостей и получать идентификаторы из API провайдера инфраструктуры.
- Дробление монолита начинается с рефакторинга ресурсов или конфигураций высокого уровня без зависимостей, затем ресурсов или конфигураций с зависимостями и завершается рефакторингом ресурсов или конфигураций низкого уровня без зависимостей.

11

Исправление ошибок

Основные темы главы:

- определение способа повторного применения изменений для восстановления функциональности;
- основные подходы к поиску проблем в IaC;
- классификация исправлений неудачных обновлений.

Позади осталось много глав, посвященных обсуждению разработки инфраструктуры как кода и совместной работы с ней. Все методы и принципы, которые вы изучили, промелькнут перед вашими глазами в критический момент, когда вы вносите изменение, оно приводит к сбою вашей системы и вам нужно откатить его! Однако IaC не поддерживает *откат* (rollback). Вы не можете полностью отменить изменения IaC до исходного состояния. Но как тогда устранять сбои, если невозможно откатывать изменения?

В этой главе основное внимание уделяется исправлению неудачных изменений IaC. Сначала мы обсудим, что значит «отмена» изменения IaC путем *наката* (roll forward) обновлений. Затем вы изучите подходы к устранению неполадок и способы исправления неудачных изменений. Хотя методы, описанные в этой главе, возможно, не применимы к каждому сценарию, с которым вы столкнетесь в вашей системе, они лежат в основе широкого набора типовых практик, которые вы можете использовать для устранения сбоев IaC.

Поиск и исправление проблем, обеспечение надежности

В этой книге я не углубляюсь в процесс и принципы устранения неполадок в вышедших из строя системах. Большая часть обсуждения способов устранения проблем сосредоточена на том, как это делать в контексте IaC. Для получения дополнительной информации об устранении неполадок и создании надежных систем я рекомендую книгу *Site Reliability Engineering* (Betsy Beyer et al.; O'Reilly, 2016).

Как правило, сначала нужно стабилизировать систему и восстановить функциональность служб и клиентов и только потом приступать к дальнейшему поиску и устранению основной причины (проблемы, которая привела к сбою). Временная «заплатка» дает вам возможность устранить неполадки и реализовать долгосрочное исправление системы.

11.1 Восстановление функциональности

Представьте, что вы работаете инженером в компании Cool Caps for Keys. Она создает индивидуальные клавиши для клавиатуры и связывает клиентов с художниками для создания клавиш на заказ. По запросу от подразделения безопасности вам необходимо сузить контроль доступа для приложений и пользователей в проектах GCP.

Вы копируете конфигурацию базы данных Google Cloud SQL и обновляете контроль доступа, чтобы обеспечить доступ с наименьшими необходимыми привилегиями для членов команды и приложений. Вы выбираете политики, необходимые для использования инфраструктуры различными приложениями, и убеждаетесь, что приложения по-прежнему работают.

Затем вы советуетесь с командой по продвижению. Их приложение обращается к базе данных напрямую, используя имя пользователя и пароль базы данных. Прямой доступ к базе данных означает, что вы можете удалить политику для `roles/cloudsql.admin` из учетной записи службы приложения, принадлежащего команде продвижения. Вы удаляете политику, тестируете изменения, вместе с командой продвижения убеждаетесь, что изменение не повлияло на работу приложения в тестовой среде, и отправляете обновление в промышленную среду, как показано на рис. 11.1.

Через час команда продвижения сообщает вам, что ее приложение продолжает выдавать ошибку о невозможности получить доступ к базе данных! Вы подозреваете, что причиной сбоя послужило ваше обновление. Первым делом вы стараетесь восстановить работу службы продвижения, чтобы она могла получить доступ к базе данных, и только потом переходите к поиску ошибки в обновлении.

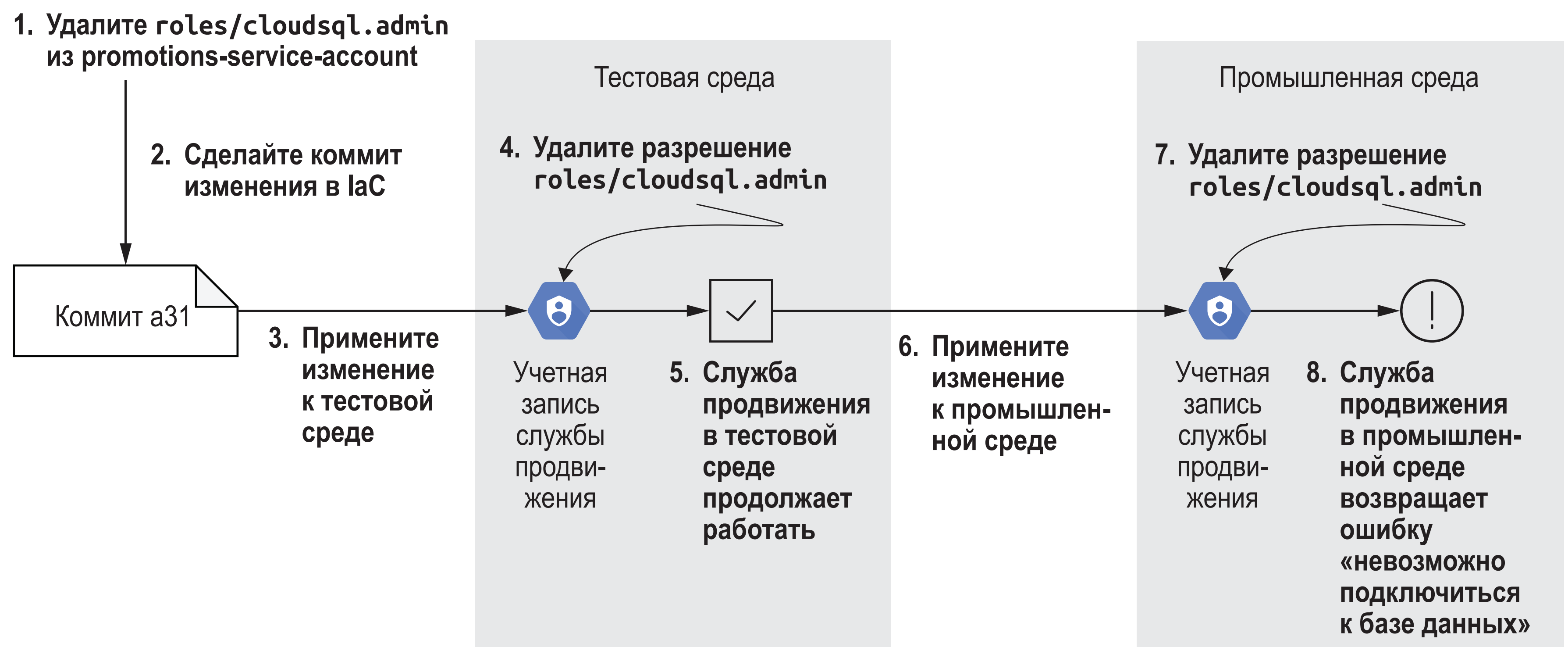


Рис. 11.1 После удаления административного доступа к базе данных для службы продвижения вы обнаружите, что это изменение лишило службу возможности доступа к базе данных

11.1.1 Отмена изменений путем наката

Вам необходимо исправить службу, чтобы пользователи могли отправлять запросы в систему. Однако просто вернуть систему в исходное рабочее состояние невозможно. Методология IaC основана на приоритете неизменности, а это означает, что любые изменения в системе, включая *отмененные* обновления, должны создавать новые ресурсы!

Например, давайте исправим службу продвижения для Cool Cars for Keys, отменив изменения и добавив роль в учетную запись службы. На рис. 11.2 показан процесс отмены коммита и добавления `roles/cloudsql.admin` обратно в учетную запись службы. Затем вы вносите изменения в тестовую и промышленную среды.



Рис. 11.2 Добавление роли администратора базы данных для службы продвижения, чтобы выполнить накат системы до рабочего состояния

Вы отменяете коммит и вносите *новые изменения* в тестовую и промышленную среды. Таким образом, вы выполняете *накат* IaC, поскольку используете неизменяемость для возврата системы в рабочее состояние.

ОПРЕДЕЛЕНИЕ *Накат* (rolling forward) IaC отменяет изменения в системе и использует принцип неизменяемости для возврата системы в рабочее состояние.

Традиционный *откат* (rollback) подразумевает возврат некоего объекта в предыдущее состояние. Но неизменность IaC означает, что на самом деле вы создаете *новое* состояние каждый раз, когда вносите изменения. Вы не можете полностью восстановить состояние инфраструктуры до прежнего состояния. Иногда вообще нет шансов восстановить инфраструктуру в предыдущее состояние, потому что ваше неудачное изменение слишком сильно нарушило систему.

Давайте обратим изменения, внесенные в учетную запись службы, и выполним обновление, чтобы вернуть разрешение на доступ команде продвижения. Прежде всего просмотрите историю коммитов, поскольку система контроля версий отслеживает все внесенные вами изменения. Коммит с префиксом a31 включает в себя удаление `roles/cloudsql.admin`:

```
$ git log --oneline -2
a3119fc (HEAD -> main) Удалить доступ администратора базы данных
у службы продвижения
6f84f5d Добавить доступ администратора базы данных службе продвижения
```

Наверняка вам хочется применить методы GitOps из главы 7, чтобы избежать внесения ручных критических изменений. Поэтому вы предпочитаете операционные изменения через IaC. Вы отменяете коммит путем отправки обновления, возвращающего службу продвижения в рабочее состояние:

```
$ git revert a3119fc
```

Вы отправляете коммит, и конвейер добавляет роль обратно в учетную запись службы. После наката обновления приложение снова работает. Вы успешно вернули состояние инфраструктуры в рабочее состояние. Однако полного возврата к исходному состоянию вы никогда не добьетесь. Вместо этого вы развернули новое состояние, более или менее совпадающее с предыдущим рабочим состоянием.

Откат IaC часто означает накат изменений в состоянии инфраструктуры. Вы используете команду `git revert`, чтобы сохранить неизменяемость и выполнить накат восстановительных обновлений в инфраструктуре.

Управление конфигурацией

Управление конфигурацией (configuration management) не отдает приоритет неизменности, но тоже выполняет накат восстановительных обновлений на сервере или ресурсе. Например, представьте, что вы установили пакет версии 3.0.0, но вам нужно вернуться к версии 2.0.0. Ваш инструмент управления конфигурацией может выбрать удаление новой версии и переустановку старой версии. Вы не восстанавливаете пакет и его конфигурацию в предыдущее состояние. Вы просто восстанавливаете сервер в новое рабочее состояние с помощью старого пакета.

11.1.2 Накат новых изменений

Преимущество подхода, основанного на накате, означает, что вы расширяете свой подход к устранению неполадок. В этом примере вы отменили неудачный коммит и восстановили функциональность службы продвижения, сопоставив новое неисправное состояние с предыдущим рабочим. Однако иногда отмена коммита не исправляет вашу систему, а только ухудшает ситуацию! Вместо этого вы накатываете *новые изменения*, которые восстанавливают функциональность.

Предположим, что сервис продвижения по-прежнему не работает после наката изменений. Вместо того чтобы пытаться исправить проблему с приложением, вы создаете новую среду с внесенными изменениями и новой службой продвижения. Вы применяете метод канареечного развертывания из главы 9, чтобы постепенно увеличивать трафик до полного восстановления приложения, как показано на рис. 11.3. После того как все запросы перейдут к новому экземпляру службы, вы отключаете неисправную среду для отладки.

IaC позволяет воспроизводить среду с меньшими усилиями. Более того, следование принципу неизменности означает, что у вас уже есть паттерн создания новой среды для изменений. Сочетание этих двух принципов помогает смягчить изменения, связанные с более высоким риском и большим потенциальным ущербом.

В ситуациях, когда мы имеем дело с данными или другими невозстановимыми ресурсами, накат изменений не позволяет перейти к предыдущему состоянию. Вы можете случайно уничтожить данные приложения или повлиять на другую инфраструктуру, устраняя каскадный сбой. Вместо наката для возврата вы можете внести новые изменения, применив методы из главы 9.

Вы также можете в той или иной степени восстановить функциональность, комбинируя возвраты и совершенно новые изменения. Расширение метода наката за счет включения новых изменений, помимо отмены старых, представляет собой полезную альтернативу для быстрого восстановления функциональности и минимизации сбоев в работе других частей вашей системы.

1. Вы обнаруживаете, что не можете исправить существующее приложение после обновления. Оно не может восстановиться

2. Разверните новый экземпляр службы продвижения

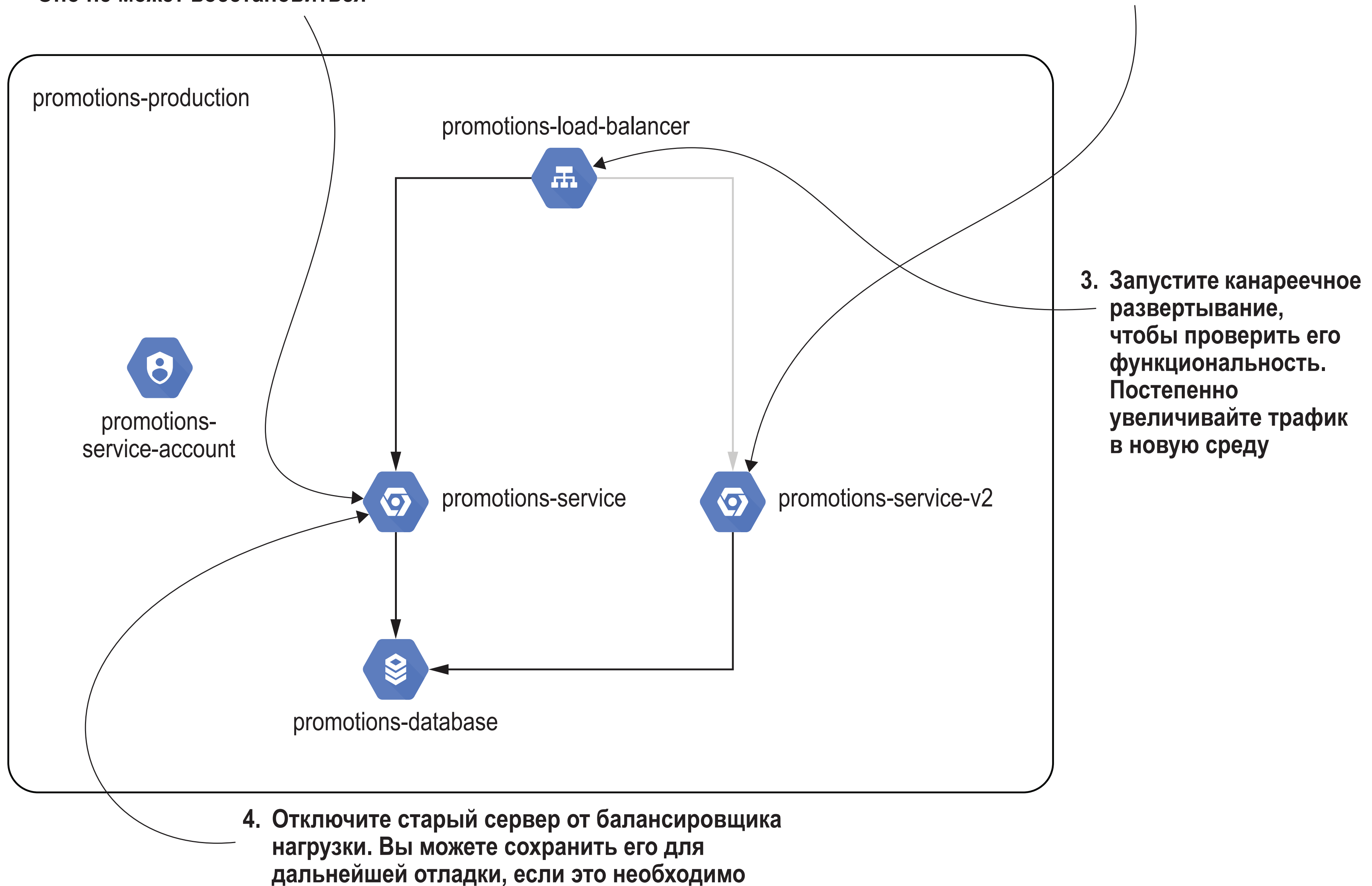


Рис. 11.3. Пока вы не можете восстановить службу продвижения, примените канареечное развертывание, чтобы переключить трафик на новый экземпляр и восстановить систему

11.2 Устранение неполадок

Итак, вам удалось временно восстановить работоспособность системы, чтобы команда по продвижению по-прежнему могла рассылать рекламные предложения в интересах Cool Caps for Keys. Однако вам все равно необходимо обезопасить IAM приложения! С чего начать, чтобы выяснить, почему служба продвижения вышла из строя, когда вы удалили административные разрешения, которые не нужны команде?

Устранение неполадок IaC следует определенным паттернам. Даже в самых сложных инфраструктурных системах многие неудачные изменения IaC обычно происходят по трем причинам: конфигурационный сдвиг, зависимости или различия окружения/среды. Проверка вашей конфигурации на наличие любой из этих причин поможет вам найти проблему и возможное ее решение.

11.2.1 Проверка наличия конфигурационного сдвига

Многие неполадки обновленной инфраструктуры возникают из-за расхождения между конфигурацией и состоянием ресурса. На

рис. 11.4 показана проверка наличия конфигурационного сдвига. Убедитесь, что конфигурация IaC для учетной записи службы соответствует реальному состоянию учетной записи службы в GCP.

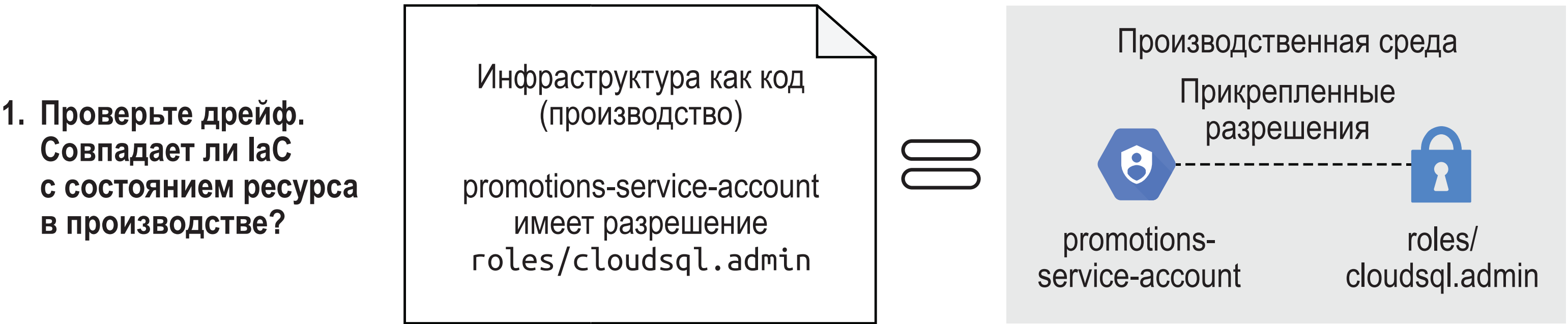


Рис. 11.4 Начните с проверки расхождения между IaC и состоянием

Проверка расхождения между кодом и состоянием гарантирует исключение любых сбоев, вызванных различиями между ними. Различия между кодом и состоянием могут привести к неожиданным проблемам. При отсутствии каких-либо различий система должна работать после обновления так, как ожидалось.

В случае с Cool Caps for Keys вы проверяете разрешения для учетной записи службы продвижения, которую вы определяете в IaC. В листинге 11.1 описывается конфигурация IaC, определяющая службу и роли.

Листинг 11.1 Учетная запись службы продвижения с правами администратора базы данных

```
from os import environ
import database
import iam
import network
import server
import json
import os

SERVICE = 'promotions'
ENVIRONMENT = 'prod'
REGION = 'us-central1'
ZONE = 'us-central1-a'
PROJECT = os.environ['CLOUDSDK_CORE_PROJECT']
role = 'roles/cloudsql.admin'

if __name__ == "__main__":
    resources = {
        'resource':
            network.Module(SERVICE, ENVIRONMENT, REGION).build() +
            iam.Module(SERVICE, ENVIRONMENT, REGION, PROJECT,
                        role).build() +
            database.Module(SERVICE, ENVIRONMENT, REGION).build() +
            server.Module(SERVICE, ENVIRONMENT, ZONE).build()
    }
```



```
with open('main.tf.json', 'w') as outfile:
    json.dump(resources, outfile,
               sort_keys=True, indent=4)
```

- ❶ Импортируем модуль базы данных для создания базы данных Google Cloud SQL.
- ❷ Импортируем модуль учетной записи службы Google и создаем конфигурацию с разрешениями.
- ❸ Импортируем сетевой модуль для построения сети и подсети Google.
- ❹ Импортируем серверный модуль для создания вычислительного экземпляра Google.
- ❺ Учетная запись службы продвижения должна иметь разрешение на роль `cloudsql.admin` для доступа к базе данных.
- ❻ Применяем модуль для создания конфигурации JSON базы данных, сети, учетной записи службы и сервера.
- ❼ Импортируем сетевой модуль для построения сети и подсети Google.
- ❽ Импортируем модуль учетной записи службы Google и создаем конфигурацию с разрешениями.
- ❾ Импортируем модуль базы данных для создания базы данных Google Cloud SQL.
- ❿ Импортируем серверный модуль для создания вычислительного экземпляра Google.
- ⓫ Записываем словарь Python в файл JSON, который будет выполнен Terraform позже.

Эквиваленты AWS и Azure

Эквивалент разрешения администратора GCP Cloud SQL в AWS аналогичен `AmazonRDSFullAccess`. Azure не имеет точного эквивалента. Вместо этого вам потребуется напрямую добавить учетную запись Azure Active Directory в базу данных и предоставить административное согласие на разрешения API базы данных SQL Azure.

Затем сравните код с разрешениями учетной записи службы продвижения в GCP. У учетной записи службы есть только разрешения `roles/cloudsql.admin`, соответствующие вашей IaC:

```
$ gcloud projects get-iam-policy $CLOUDSDK_CORE_PROJECT
bindings:
- members:
- serviceAccount:promotions-prod@infrastructure-as-code-book
➡.iam.gserviceaccount.com
role: roles/cloudsql.admin
version: 1
```

Если вы обнаружите расхождение конфигурации между IaC и активным состоянием ресурса, вы можете дополнительно выяснить, влияет ли оно на функциональность системы. Вы можете устранить некоторые отклонения, чтобы убедиться, что они не способствуют устранению основной причины. Однако тот факт, что вы обнару-

жили конфигурационный сдвиг, еще не означает, что система обязательно неисправна! Некоторые расхождения никак не влияют на работоспособность.

11.2.2 Проверка зависимостей

Если вы пришли к выводу, что конфигурационный сдвиг не является причиной сбоя, попробуйте проверить ресурсы, которые зависят от вашего обновленного файла. На рис. 11.5 отображены ресурсы, зависящие от учетной записи службы. Как в IaC, так и в производственной среде сервер зависит от учетной записи службы.

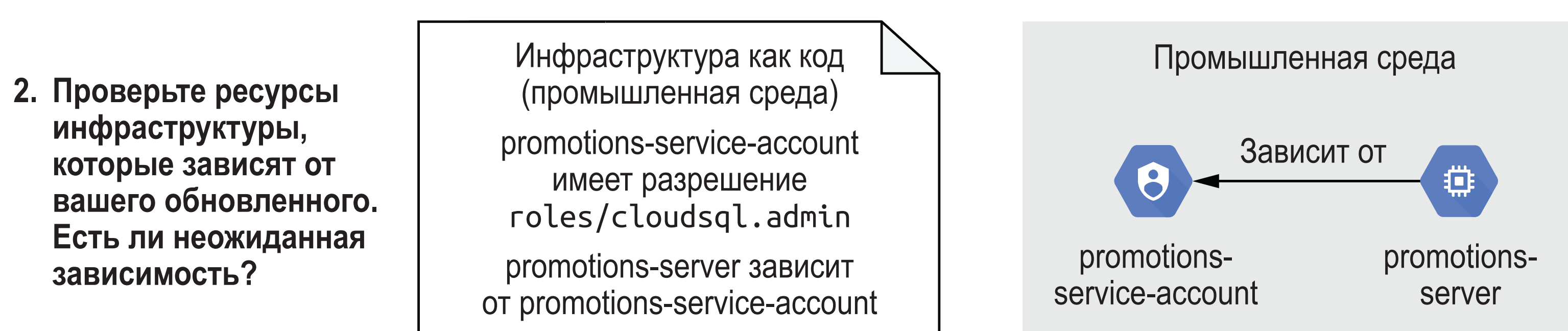


Рис. 11.5 Устранение неполадок ресурсов, которые зависят от обновленного вами ресурса

Вам нужно проверить, соответствуют ли ожидаемые зависимости фактическим. Неожиданные зависимости искажают эффект от обновления. Просматривая код в листинге 11.2, вы убедитесь, что электронная почта учетной записи службы передается на сервер.

Листинг 11.2 Сервер службы продвижения зависит от учетной записи службы

```
class Module():
    def __init__(self, service, environment,
                  zone, machine_type='e2-micro'):
        self._name = f'{service}-{environment}'
        self._environment = environment
        self._zone = zone
        self._machine_type = machine_type

    def build(self):
        return [
            {
                'google_compute_instance': {
                    self._environment: {
                        'allow_stopping_for_update': True,
                        'boot_disk': [{
                            'initialize_params': [{
                                'image': 'ubuntu-1804-lts'
                            }]
                        }],
                    },
                    'machine_type': self._machine_type
                }
            }
        ]
```



```

        'name': self._name,
        'zone': self._zone,
        'network_interface': [{
            'subnetwork':
                '${google_compute_subnetwork.' +
                f'{self._environment}' + '.name}',
            'access_config': {
                'network_tier': 'STANDARD'
            }
        }],
        'service_account': [{
            'email': '${google_service_account.' +
            f'{self._environment}' + '.email}',
            'scopes': ['cloud-platform']
        }]
    }
}
]

```

- ❶ Применяем модуль для создания конфигурации JSON сервера.
- ❷ Создаем вычислительный экземпляр Google с использованием ресурса Terraform на основе имени, адреса, региона и сети.
- ❸ Фабрика сервера службы продвижения использует учетную запись службы для доступа к службам GCP.

Эквиваленты AWS и Azure

Создайте сеть в AWS или Azure. Затем обновите код, чтобы использовать ресурс Terraform виртуальной машины Azure Linux (<http://mng.bz/J22p>) с управляемым блоком identity. Блок identity должен включать список идентификаторов пользователей с разрешениями на доступ к Azure. Для AWS вам необходимо определить профиль экземпляра IAM для ресурса AWS EC2 в Terraform.

Однако команда по продвижению отмечает, что ее приложение *напрямую* обращается к базе данных, используя ее IP-адрес, имя пользователя и пароль. Зачем серверу нужна учетная запись службы, если приложение считывает строку для подключения к базе данных из файла?

Вы понимаете, что это явное несоответствие между ожидаемой логикой работы приложения и конфигурацией. Вы просите команду продвижения показать вам код приложения. В конфигурации приложения не используются IP-адрес базы данных, имя пользователя и пароль!

После дополнительной отладки с командой продвижения вы обнаруживаете, что их приложение подключается к базе данных на локальном хосте. В конфигурации используется прокси-сервер Cloud SQL Auth (<https://cloud.google.com/sql/docs/mysql/sql-proxy>), который обрабатывает соединение и осуществляет вход в базу данных!

Поэтому учетной записи службы, подключенной к серверу, требуется доступ к базе данных.

На рис. 11.6 показано, что приложение обращается к базе данных через прокси. Прокси-сервер использует учетную запись службы для аутентификации и доступа к базе данных. Учетной записи службы необходим доступ к базе данных с помощью политики.

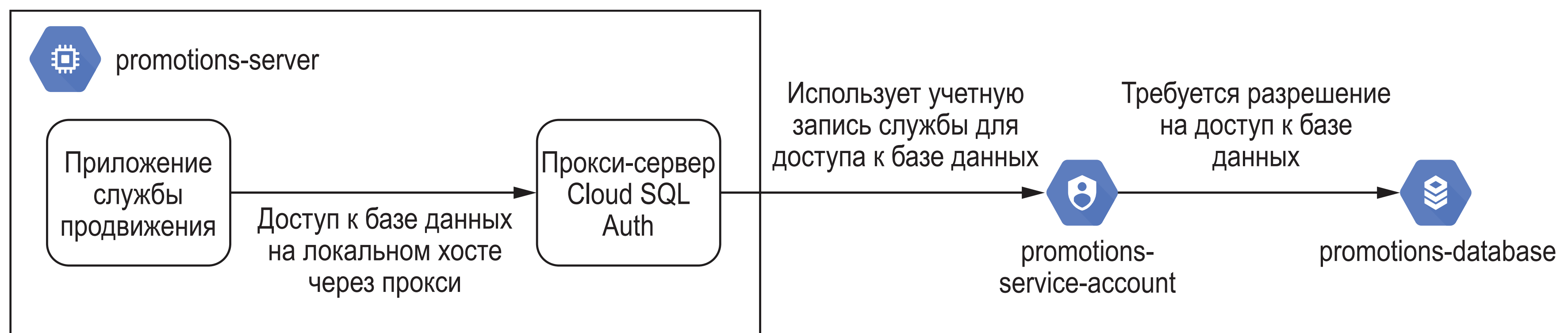


Рис. 11.6 Приложение обращается к базе данных через прокси-сервер, для которого требуется учетная запись службы с разрешениями базы данных

Эквиваленты AWS и Azure

Эквивалентом AWS для прокси-сервера GCP Cloud SQL Auth является прокси-сервер Amazon RDS. Прокси-сервер помогает обеспечить подключение к базе данных и позволяет избежать необходимости указывать имена пользователей и пароли базы данных в коде приложения.

Azure не имеет эквивалентного варианта прокси-сервера SQL. Вместо этого вам необходимо настроить частную ссылку Azure на базу данных. При этом IP-адрес будет выделен в частной сети по вашему выбору. Вы можете настроить свою базу данных, чтобы разрешить приложению вход в систему с помощью субъекта службы Azure Active Directory.

Поздравляю, вы узнали, почему приложение перестало работать, когда вы удалили учетную запись службы! Но у вас возникли кое-какие подозрения. Разве вы не должны были обнаружить ту же проблему в среде тестирования? В конце концов, вы протестировали изменение в тестовой среде, и приложение там работало нормально.

11.2.3 Проверка различий между средами

Почему изменение сработало при тестировании, но не в промышленной среде? Когда вы проверяли приложение в ходе тестирования, оно не подключалось к базе данных на локальном хосте. Вместо этого оно использовало IP-адрес базы данных, имя пользователя и пароль.

Вы объясняете команде продвижения, что промышленная конфигурация IaC использует прокси-сервер Cloud SQL Auth, а тестируемая конфигурация IaC напрямую обращается к базе данных (рис. 11.7). Обе конфигурации используют разрешение `roles/cloudsql.admin`.

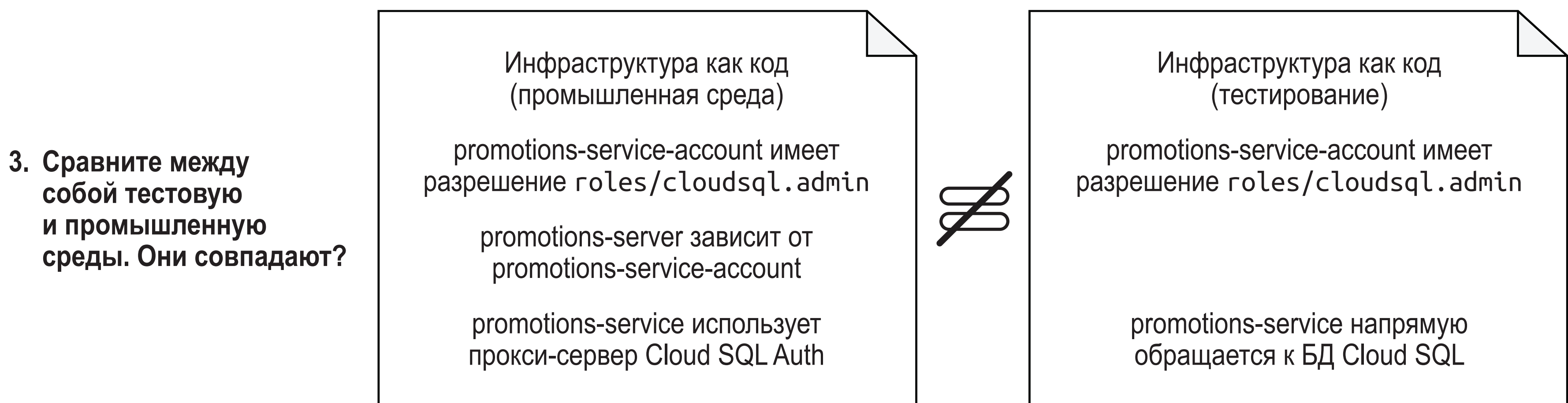


Рис. 11.7 Проверьте наличие различий между тестовой и промышленной средами, чтобы понять, почему некоторые обновления не работают в промышленной среде

После дальнейшего обсуждения с командой по продвижению вы обнаруживаете, что команда внедрила срочное изменение для обеспечения безопасности промышленной среды с помощью прокси-сервера Cloud SQL Auth. Однако у команды не было возможности обновить тестовую среду! Несовпадение сред привело к тому, что ваше обновление успешно прошло проверку в тестовой среде, но вызвало сбой в промышленной.

Вам нужно, чтобы тестовая и промышленная среды были как можно более похожими. Однако вы не всегда можете воспроизвести промышленную среду в тестовой. Как результат этого вы столкнетесь с неудачными обновлениями из-за несоответствий в той или иной среде. Систематическое сравнение двух сред помогает выявить пробелы в процессах тестирования и развертывания обновлений.

Хотя парадигма IaC предусматривает документирование всех изменений и настроек вашей системы, вы все равно можете столкнуться с сюрпризами при переходе между средами. На рис. 11.8 обобщен структурированный подход к отладке неудачных изменений в IaC из нашего примера службы продвижения. Вы проверяете наличие конфигурационного сдвига, зависимости и, наконец, различия между средами.

Итак, вы нашли истинную причину сбоя и наконец можете реализовать окончательное решение проблемы. Теперь вам необходимо устранить расхождение между средами и вернуться к доступу с наименьшими привилегиями, чтобы защитить учетную запись службы продвижения.

Упражнение 11.1

Команда сообщает, что ее приложение больше не может подключаться к другому приложению. Приложение работало на прошлой неделе, но с понедельника запросы не выполняются. Команда не вносила никаких изменений в свое приложение и подозревает, что проблема кроется в правиле брандмауэра. Какие шаги вы можете предпринять для устранения проблемы (выберите все подходящие варианты):

- a** войдите в панель управления аккаунтом облачного провайдера и проверьте правила брандмауэра для приложения;
 - b** разверните новую инфраструктуру и приложения в «зеленой» среде для тестирования;
 - c** изучите изменения в IaC для приложения;
 - d** сравните правила брандмауэра у облачного провайдера с IaC;
 - e** отредактируйте правила брандмауэра и разрешите весь трафик между приложениями.
- Ответы на упражнения даны в приложении В.

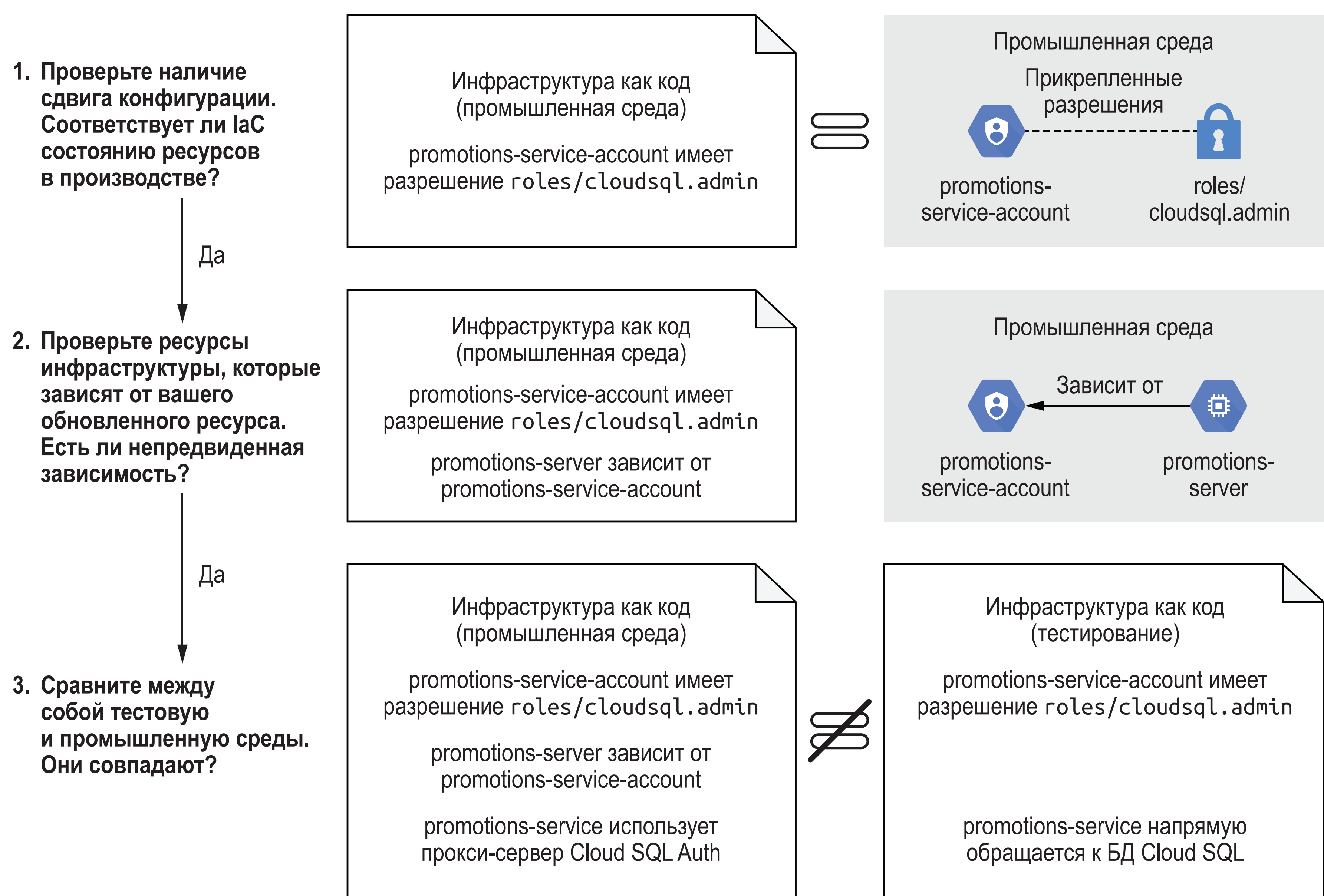


Рис. 11.8 Для устранения проблем с неудачным обновлением вы последовательно проверяете отклонения, неожиданные зависимости и различия между тестовой и промышленной средами

11.3 Устранение причины сбоя

Ваша первоначальная задача заключалась в обновлении разрешений учетной записи службы для каждого приложения компании Cool Cars for Keys, чтобы обеспечить доступ к службам с минимальными привилегиями. Вы попытались удалить административный доступ к базе данных из учетной записи службы продвижения, но это не удалось. Вы нашли причину сбоя и теперь можете ее устранить.

На этом этапе вы можете почувствовать легкое нетерпение! Ведь вы еще не закончили обновлять доступ для других приложений в Cool Caps for Keys. Однако не стоит спешить и менять все сразу. Внесение множественных изменений может затруднить поиск причины сбоя (об этом говорилось в главе 7). Ваша тестовая среда по-прежнему отличается от промышленной, и вы все равно можете повлиять на приложение службы продвижения, если внесете слишком много изменений одновременно.

На протяжении всей книги я рекомендую вносить незначительные обновления, чтобы минимизировать потенциальный ущерб. Аналогичным образом инкрементные исправления разбивают на части изменения, которые необходимо внести в систему, чтобы предотвратить будущие сбои.

ОПРЕДЕЛЕНИЕ В соответствии с методом *инкрементных исправлений* (incremental fixes) изменения разбивают на небольшие шаги, чтобы постепенно улучшить систему и предотвратить сбои в будущем.

Внесение небольших изменений в конфигурацию и постепенное их развертывание поможет вам распознать первые признаки проблемы и подготовить IaC для будущего успешного обновления.

11.3.1 Устранение расхождений

Как я упоминала в главе 2, любые ручные изменения состояния инфраструктуры необходимо синхронизировать с IaC. Если вы обнаружите расхождение, первым делом нужно его устранить! Если вы действительно делаете ставку на IaC, в вашей системе не должно быть слишком много критических изменений.

Напомню, что в конфигурацию службы продвижения Cool Caps for Keys было внесено кардинальное изменение, которое привело к различию между тестовой и промышленной средами. Рабочее приложение использует прокси-сервер Cloud SQL Auth для подключения к базе данных, а тестовое приложение подключается к базе данных напрямую через IP-адрес и пароль. Вам необходимо создать прокси-сервер Cloud SQL Auth в тестовой среде.

Чтобы исправить расхождение, нужно восстановить текущее состояние инфраструктуры в конфигурации. На рис. 11.9 показана реконструкция установки прокси-сервера Cloud SQL Auth по аналогии с промышленным сервером. Затем вы добавляете команды в сценарий IaC и применяете их к среде тестирования.

В этом примере команда продвижения не обновила конфигурацию IaC после срочного ручного обновления. В результате вам пришлось тратить время на поиск причины и установку прокси-сервера Cloud SQL Auth. Незарегистрированное внештатное изменение, та-

кое как добавление прокси-сервера, привело к сбою последующего обновления, исправление которого требует еще больше времени и усилий.

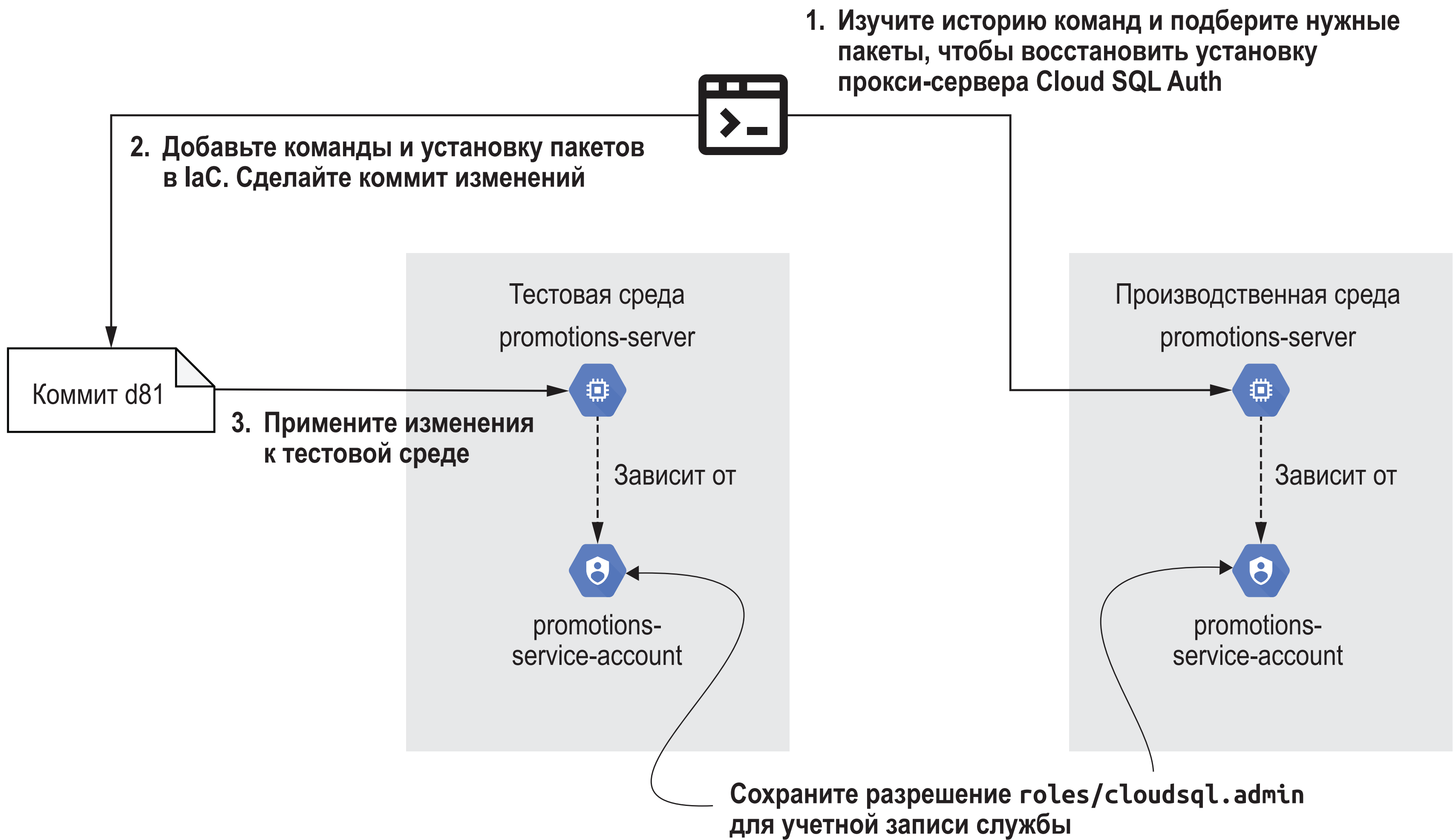


Рис. 11.9 Вам необходимо установить в тестовой среде пакет прокси Cloud SQL Auth на сервер службы продвижения, чтобы устранить расхождение, возникающее после обновления

Чтобы избежать подобных проблем, используйте процесс миграции на IaC, описанный в главе 2. Обновление конфигурации IaC после внесения срочных ручных изменений помогает свести к минимуму различия между средами и конфигурационный сдвиг между IaC и фактическим состоянием. Если вам нужно восстановить состояние инфраструктуры, помните, что глава 2 содержит общий пример миграции существующей инфраструктуры в IaC. Однако обычно приходится находить готовый или разрабатывать свой инструмент для преобразования состояния в IaC.

Давайте напишем сценарий IaC для установки прокси. Вы изучили историю команд на рабочем сервере службы продвижения и восстановили состояние прокси-сервера Cloud SQL Auth. Код в листинге 11.3 демонстрирует автоматизацию команд и процесса установки в сценарии запуска сервера службы продвижения.

Листинг 11.3 Установка прокси-сервера Cloud SQL Auth в сценарии запуска сервера

```
class Module():
    def _startup_script(self):
        proxy_download = 'https://dl.google.com/cloudsql/' + \
            'cloud_sql_proxy.linux.amd64'
```

1
2
2


```

exec_start = '/usr/local/bin/cloud_sql_proxy ' + \
    '-instances=${google_sql_database_instance.' + \
    f'{self._environment}.connection_name}}=tcp:3306'

return f"""
#!/bin/bash
wget {proxy_download} -O /usr/local/bin/cloud_sql_proxy
chmod +x /usr/local/bin/cloud_sql_proxy

cat << EOF > /usr/lib/systemd/system/cloudsqlproxy.service
[Install]
WantedBy=multi-user.target

[Unit]
Description=Google Cloud Compute Engine SQL Proxy
Requires=networking.service
After=networking.service

[Service]
Type=simple
WorkingDirectory=/usr/local/bin
ExecStart={exec_start}
Restart=always
StandardOutput=journal
User=root
EOF

systemctl daemon-reload
systemctl start cloudsqlproxy
"""

def build(self):
    return [
        {
            'google_compute_instance': {
                self._environment: {
                    'metadata_startup_script': self._startup_script()
                }
            }
        }
    ]

```

- ❶ Создаем сценарий запуска, который восстанавливает команды ручной установки для прокси-сервера Cloud SQL Auth.
- ❷ Задаем переменную в качестве адреса загрузки прокси-сервера.
- ❸ Задаем переменную, которая запускает двоичный файл прокси-сервера Cloud SQL Auth на порту 3306.
- ❹ Возвращаем сценарий оболочки, который устанавливает прокси-сервер и запускает его вместе с сервером.
- ❺ Настраиваем демон systemd для запуска и остановки прокси-сервера Cloud SQL Auth.
- ❻ Применяем модуль для создания конфигурации JSON вычислительного экземпляра Google и добавляем сценарий запуска для установки прокси-сервера.
- ❼ Добавляем сценарий запуска (startup script) на сервер. Другие атрибуты опущены для ясности.

Эквиваленты AWS и Azure

В случае AWS и Azure вам не нужно устанавливать программное обеспечение для прокси-сервера на экземпляре. Если вы хотите воспроизвести код из листинга 11.3 в AWS и Azure для тренировки, вы можете передать сценарий запуска ресурсу как `user_data` в экземпляр AWS или `custom_data` в виртуальную машину Azure Linux.

Вы *не* обновляете учетную запись службы новыми разрешениями! В соответствии с принципом инкрементных исправлений вам следует избегать добавления дополнительных изменений, которые нужно отслеживать по мере перехода к рабочей версии. Вы добавляете сценарий запуска на сервер службы продвижения и меняете тестовую среду без дополнительных обновлений.

Сценарий запуска, менеджер конфигурации или сборщик образов?

В этом примере я использую поле сценария запуска, чтобы не усложнять синтаксис. Рекомендую вам реализовать настройку любых новых пакетов или служб с помощью менеджера конфигурации или системы сборки образов.

Например, диспетчер конфигурации может перенести процесс установки прокси-сервера Cloud SQL Auth на любой сервер с приложением службы продвижения. Аналогично конструктор образов настраивает прокси-сервер для каждого создаваемого образа! Всякий раз, когда вы используете образ приложения для службы продвижения, в сервер будет встроен прокси-сервер.

11.3.2 Согласование различий в средах

Хотя вы обновили IaC, чтобы учесть расхождения, вам также необходимо убедиться, что промышленная и тестовая среды используют новую конфигурацию IaC. В нашем примере с Cool Caps for Keys вы должны убедиться, что соединение с базой данных работает в тестовой среде. Затем вы просите команду по продвижению обновить конфигурацию своего приложения для подключения к базе данных через прокси-сервер на локальном хосте.

Команда по продвижению разворачивает конфигурацию своего приложения для использования Cloud SQL Auth через прокси-сервер в среде тестирования, запускает тесты и обновляет промышленную среду, как показано на рис. 11.10. Вы сохраняете разрешение `roles/cloudsql.admin` для учетной записи службы, поскольку оно необходимо прокси-серверу.

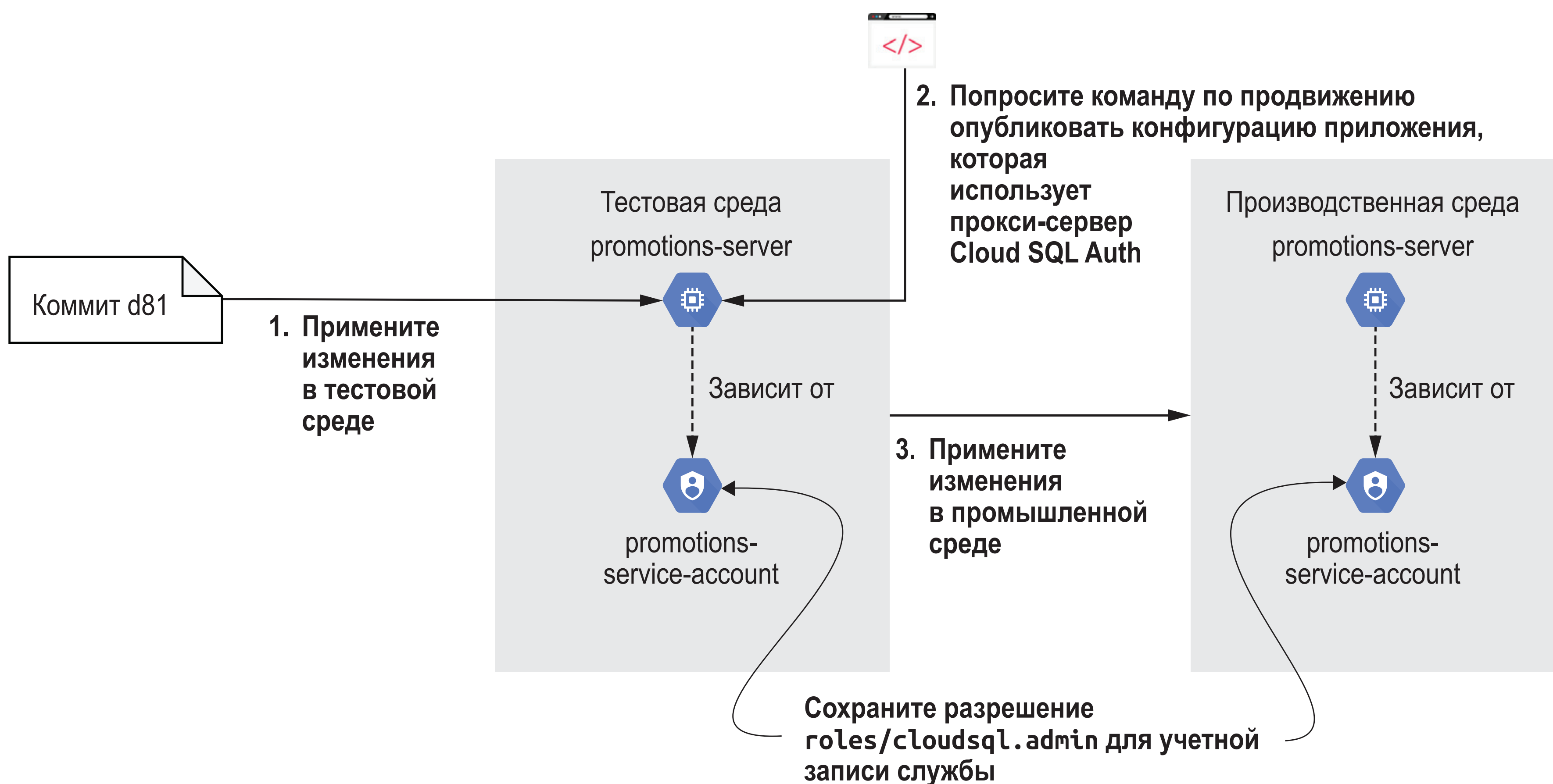


Рис. 11.10 Вам необходимо отправить изменения IaC на сервер службы продвижения как в тестовой, так и в промышленной среде

После обновления промышленный сервер будет создан заново с новым сценарием запуска. После дополнительного комплексного тестирования приложения вы убеждаетесь, что обе среды успешно обновлены.

Зачем начинать с исправления конфигурационного сдвига, прежде чем устранять различия между промышленной и тестовой средами? В этом примере вы сначала убираете сдвиг, чтобы не тратить много времени на установку пакетов вручную в тестовой среде. И если вы обновите IaC и автоматизируете установку пакетов, то можете убедиться, что обновление работает в тестовой среде, прежде чем отправлять его в промышленную среду.

С другой стороны, вы можете сначала устранить большое расхождение между средами. В таком случае максимально синхронизируйте тестовую и промышленную среды, прежде чем устранять конфигурационный сдвиг, потому что вам нужна точная тестовая среда.

Устранение расхождений между средами и конфигурационного сдвига поможет другим людям вносить обновления в систему. Им не придется беспокоиться о том, чтобы найти разницу в конфигурации или настроить прокси вручную. Время, которое вы потратите на обновление IaC, поможет вам и вашей компании избежать дополнительных трудозатрат на отладку!

11.3.3 Внесение изначально необходимого изменения

Вы свели к минимуму потенциальный ущерб от сбоя, устранив сдвиг и обновив среду. Теперь, наконец, можно вносить запланированные изменения. Отладка и инкрементные исправления меняют вашу инфраструктуру. Когда вы вернетесь к реализации изначально-

ного изменения, вам может потребоваться скорректировать код конфигурации.

Давайте завершим запланированное изменение приложения по продвижению компании Cool Caps for Keys. Напомним, что служба безопасности попросила вас удалить административные разрешения из учетной записи службы. Вы должны настроить доступ с минимальными привилегиями с учетом использования прокси-сервера Cloud SQL Auth.

Вы знаете, что аккаунт службы должен иметь доступ к базе данных, поскольку приложение использует прокси-сервер Cloud SQL Auth. Теперь нужно выяснить, какой минимальный доступ должно использовать приложение. Разрешение `roles/cloudsql.client` обеспечивает достаточный доступ, чтобы получить список экземпляров и подключиться к ним.

На рис. 11.11 показано, как вы меняете права учетной записи службы с административного доступа на `roles/cloudsql.client`. Вы отправляете это изменение в тестовую среду, убеждаетесь, что приложение по-прежнему работает, и развертываете разрешение `roles/cloudsql.client` в промышленной среде.

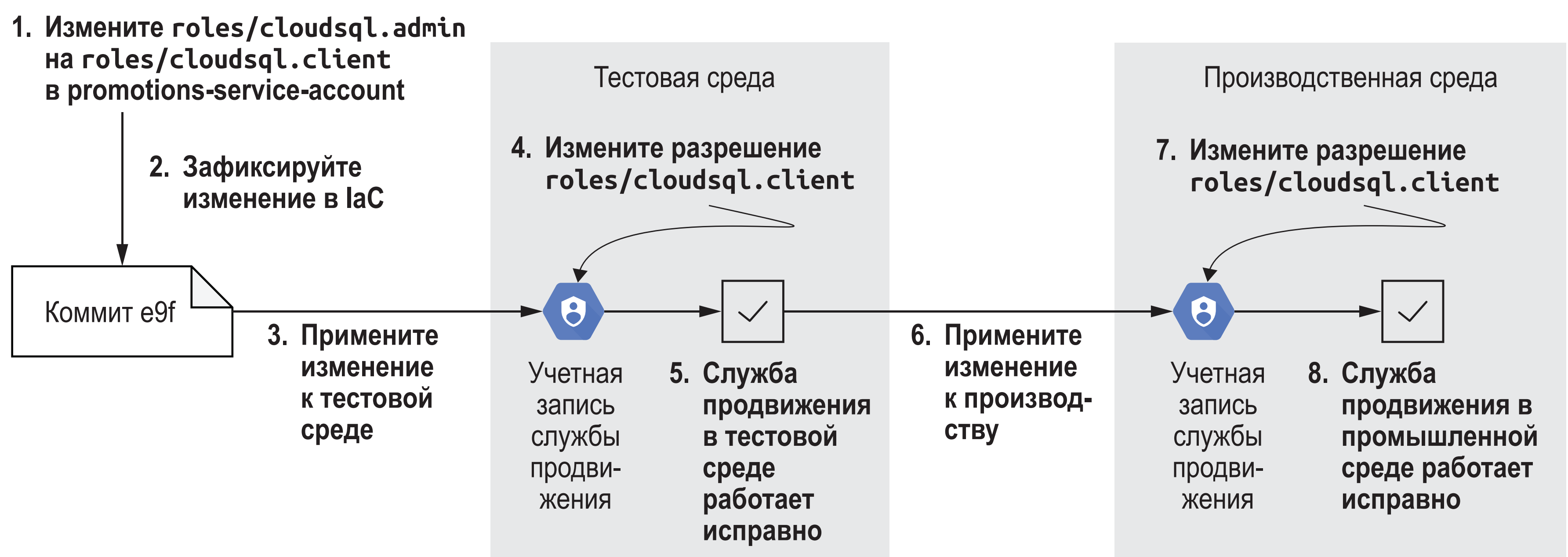


Рис. 11.11 Вам необходимо отправить изменения IaC на сервер службы продвижения в тестовой и промышленной средах

Вы устранили разницу между тестовой и промышленной средами, связанную с использованием прокси-сервера. Теоретически тестовая среда теперь должна выявлять любые проблемы с вашими изменениями. В листинге 11.4 мы меняем права доступа учетной записи службы с `roles/cloudsql.admin` на `roles/cloudsql.client`.

Листинг 11.4 Изменение роли учетной записи службы с администратора на клиента

```

from os import environ
import database
import iam

```



```

import network
import server
import json
import os

SERVICE = 'promotions'
ENVIRONMENT = 'prod'
REGION = 'us-central1'
ZONE = 'us-central1-a'
PROJECT = os.environ['CLOUDSDK_CORE_PROJECT']
role = 'roles/cloudsql.client' ❶

if __name__ == "__main__":
    resources = { ❷
        'resource': ❷
        network.Module(SERVICE, ENVIRONMENT, REGION).build() + ❷
        iam.Module(SERVICE, ENVIRONMENT, REGION, PROJECT, ❸
            role).build() + ❸
        database.Module(SERVICE, ENVIRONMENT, REGION).build() + ❷
        server.Module(SERVICE, ENVIRONMENT, ZONE).build() ❷
    }

    with open('main.tf.json', 'w') as outfile:
        json.dump(resources, outfile,
            sort_keys=True, indent=4)

```

- ❶ Изменяем роль учетной записи службы продвижения с администратора на клиента, что позволяет подключаться к экземпляру базы данных.
- ❷ Импортируем модули сети, базы данных и сервера без изменения ресурсов.
- ❸ Импортируем учетную запись службы и присоединяем к ее разрешениям роль `roles/cloudsql.client`.

Эквиваленты AWS и Azure

Эквивалент прав администратора GCP Cloud SQL в AWS аналогичен `AmazonRDSFullAccess`. Azure не имеет точного эквивалента. Вместо этого вам потребуется напрямую добавить учетную запись Azure Active Directory в базу данных и предоставить административное согласие на разрешения для API базы данных SQL Azure.

В AWS вы можете добавить действие `gds-db:connect` к роли IAM, прикрепленной к экземпляру EC2. В Azure вам потребуется отозвать административный доступ и предоставить доступ `SELECT` пользователю Azure AD, связанному с пользователем базы данных (<http://mng.bz/woo7>).

Вы делаете коммит и применяете обновление. Тестовая среда проверяет его и подтверждает, что приложение по-прежнему работает! Вы убеждаетесь в этом вместе с командой по продвижению, которая одобряет развертывание обновления в мысленной среде.

Команда накатывает изменения разрешений в промышленную среду, проводит сквозное тестирование и подтверждает, что при-

ложение может получить доступ к базе данных! После нескольких недель отладки и внесения изменений вы, наконец, сможете исправить другие приложения в Cool Cars for Keys.

Зачем было посвящать целую главу одному примеру исправления неудачных обновлений? Он хорошо отражает реальность исправления сбоев IaC. Вы должны устранить неполадку как можно быстрее, не усугубляя ее.

Накат изменений помогает восстановить рабочее состояние системы и свести к минимуму сбои в работе ресурсов инфраструктуры. Затем вы можете заняться устранением основной причины. Многие сбои инфраструктуры происходят из-за накопившегося конфигурационного сдвига, зависимостей или различий между тестовой и промышленной средами. После устранения всех этих несоответствий вы можете вносить в систему требуемое обновление.

Обучение искусству наката изменений в IaC требует времени и опыта. Хотя вы можете просто войти в консоль облачного провайдера и внести изменения вручную, чтобы система заработала, помните, что такая заплатка очень быстро отвалится и не поможет долгосрочному восстановлению системы. Использование IaC для постоянного отслеживания и инкрементного исправления системы сводит к минимуму необходимость ремонта и предоставляет контекст для всех, кто обновляет систему после вас.

Краткое содержание главы

- Устранение сбоев IaC подразумевает накат исправлений, а не откат обновлений.
- Накат исправлений IaC для возврата системы в рабочее состояние основан на принципе неизменности.
- Прежде чем заниматься отладкой и внесением долгосрочных исправлений, стабилизируйте систему и приведите ее в рабочее состояние.
- При устранении неполадок IaC проверяйте наличие отклонений, неожиданных зависимостей между компонентами и различий между средами как частей основной причины сбоя.
- Выполняйте инкрементные исправления, чтобы быстро распознавать угрозу сбоя и уменьшать потенциальный ущерб.
- Прежде чем повторно разворачивать обновление, которое не удалось, убедитесь, что вы устранили конфигурационный сдвиг и различия между средами для точного тестирования и будущих обновлений системы.
- Реконструкция состояния IaC с целью устранения отклонений допускает применение ручных манипуляций для настройки сервера или преобразование метаданных инфраструктуры в IaC.

Стоимость облачных услуг

Основные темы главы:

- исследование факторов, влияющих на стоимость облачных услуг;
- сравнение методов оптимизации затрат;
- применение тестов для разработки экономичной IaC;
- расчет сметы стоимости инфраструктуры.

Услуги облачных провайдеров привлекают и восхищают нас простотой получения ресурсов. Ведь создать ресурс можно одним щелчком мыши или одной строкой кода. Увы, по мере развития инфраструктуры организации возрастает и стоимость облачных вычислений – и эту проблему нельзя игнорировать. Некоторые обновления IaC существенно влияют на общую стоимость облачных услуг!

Вы должны заранее учитывать затраты на инфраструктуру точно так же, как учитываете вопросы безопасности. Если вы создадите инфраструктуру и обнаружите, что превысили допустимый бюджет, то в попытках удалить ресурсы и сократить расходы на облачные вычисления вы можете нарушить работу системы. В главе 8 я рекомендовала последовательно придерживаться рецептов безопасности, как если бы вы хотели испечь торт. Стоимость ингредиентов влияет на то, *сколько* тортов вы сможете испечь, и это важно знать, *прежде чем начинать*.

В этой главе рассматриваются методы, которые можно объединить с IaC для управления затратами на облачные вычисления и сокращения неиспользуемых ресурсов. Вы найдете здесь описание некоторых приемов управления затратами, которые я рассматриваю в контексте IaC. Применение этих методов не ограничивается первичным развертыванием инфраструктуры. Регулярно возвращайтесь к ним для повторной оптимизации затрат по мере развития вашей системы в зависимости от спроса клиентов, масштаба организации и величины счетов от провайдеров облачных услуг.

А что насчет затрат на центр обработки данных или управляемые услуги?

Я уделяю основное внимание облачным вычислениям из-за их гибкости и возможности выставления счетов по требованию. Стоимость вычислений в центре обработки данных обычно учитывают с помощью системы начисления платежей вашей организации. Каждое бизнес-подразделение устанавливает бюджет для ресурсов своего центра обработки данных, а технологический отдел начисляет ежемесячную сумму к оплате в зависимости от фактического использования ресурсов, принимая во внимание эксплуатационные расходы центра обработки данных.

Вы всегда можете применить методы управления факторами затрат с помощью контроля и оценки затрат. Однако методы экономии и оптимизации, которые я описываю, не обязательно сработают в вашей конкретной ситуации (независимо от того, используете вы облако, центр обработки данных или управляемые услуги). В зависимости от масштаба, географической архитектуры, бизнес-сферы или наличия центра обработки данных в компании ваши варианты использования и системы могут потребовать специализированной оценки либо изменения платформы.

12.1 Управление факторами затрат

Предположим, вы работаете консультантом в компании, которой необходимо перенести свою платформу, поддерживающую онлайн-конференции и мероприятия, в общедоступное облако. Компания просит вас «перенести» ее конфигурацию из своего дата-центра в облако. Вы помогаете командам компании создавать инфраструктуру в GCP, применяя все принципы и методы, о которых вы узнали из этой книги. В итоге компания развертывает платформу на GCP и успешно обслуживает своего первого клиента: небольшую трехчасовую конференцию сообщества.

Через несколько недель после мероприятия ваш клиент внезапно назначает встречу в офисе. На встрече клиент показывает вам счет за облако от провайдера. Компания потратила более \$10 000 на за-

пуск и поддержку одной трехчасовой конференции! Финансовый отдел, похоже, недоволен расходами, тем более что компания потеряла деньги на проведении конференции. Перед вами стоит следующая задача: максимально снизить стоимость конференции.

Реальны ли расходы, о которых мы говорим?

В предыдущем примере упоминается вымышленный, очень упрощенный счет за облако, который приблизительно соответствует стоимости услуги конференц-платформы на основе калькулятора цен GCP (<https://cloud.google.com/products/calculator>) по состоянию на 2021 год. Оценки могут учитывать не все нюансы необходимых вам предложений, обновленные тарифы платформ, различия между средами или размерами, которые вы можете использовать в сопоставимой системе. Я округлила промежуточные итоги, чтобы упростить пример.

Если вы запустите этот пример, вы можете достичь квоты GCP для экземпляров машин типа N2D. Серверы превысят уровень бесплатного пользования платформой! Вы можете изменить тип машины на экземпляры уровня бесплатного пользования, чтобы запускать примеры бесплатно.

Благодаря методам тегирования, которые вы позаимствовали из главы 8, можно проанализировать счет и определить, какие ресурсы были затребованы конференцией и ее средой. Допустим, вам удалось разбить свой счет за облачные вычисления, как показано в табл. 12.1, и выделить затраты по типу и размеру инфраструктурного ресурса.

Таблица 12.1. Расходы на облачные услуги в зависимости от предложения и среды

Услуга	Промежуточный итог, долл., тестовая среда	Промежуточный итог, долл., производственная среда	Промежуточный итог
Вычисления (серверы)	400	3600	4000
База данных (Cloud SQL)	250	2250	2500
Обмен сообщениями (Pub/Sub)	100	900	1000
Объектное хранилище (Cloud Storage)	100	900	1000
Передача данных (сетевой трафик)	100	900	1000
Прочее (Cloud CDN, поддержка)	50	450	500
Итог	1000	9000	10 000

Эквиваленты AWS и Azure

Счет за облачные услуги обычно содержит конкретные названия эквивалентных сервисов Azure и AWS. Для ясности я перечисляю некоторые предложения GCP, приближенные к предложениям AWS или Azure:

- база данных (Cloud SQL) – Amazon RDS, база данных SQL Azure;
- обмен сообщениями (Pub/Sub) – Amazon Simple Queue Service (SQS) и Simple Notification Service (SNS), Azure Service Bus;
- объектное хранилище (облачное хранилище) – Amazon S3, Azure Blob Storage;
- прочее (Cloud CDN, поддержка) – Amazon CloudFront, Azure Content Delivery Network (CDN).

Разделение затрат по предложениям и средам помогает вам определить, какие факторы влияют на стоимость и где вам следует провести более углубленное исследование. Чтобы приступить к сокращению счетов, вы должны определить *источники затрат* – факторы или действия, которые влияют на общую стоимость.

ОПРЕДЕЛЕНИЕ *Источники затрат* (cost driver) – это факторы или действия, которые влияют на общую стоимость облачных вычислений.

Для оценки источников затрат рассчитайте процентную долю стоимости облачных услуг. Некоторые услуги всегда будут стоить дороже остальных. Вы по-прежнему можете использовать разбивку с помощью тегов, чтобы определить объекты для оптимизации. Разбивка затрат по средам помогает вам определить соотношение расходов между средами тестирования и производства. Сравнение этих двух сред даст вам более четкое представление о том, в какой среде есть неэффективные операции, которые вы можете оптимизировать.

На основе вашей разбивки вы рассчитываете долю затрат для каждой услуги и среды. На рис. 12.1 видно, что вычислительные ресурсы дают 40 % от всех затрат. Вы также видите, что команда потратила 10 % от общей суммы на тестовую среду и 90 % на производственную.

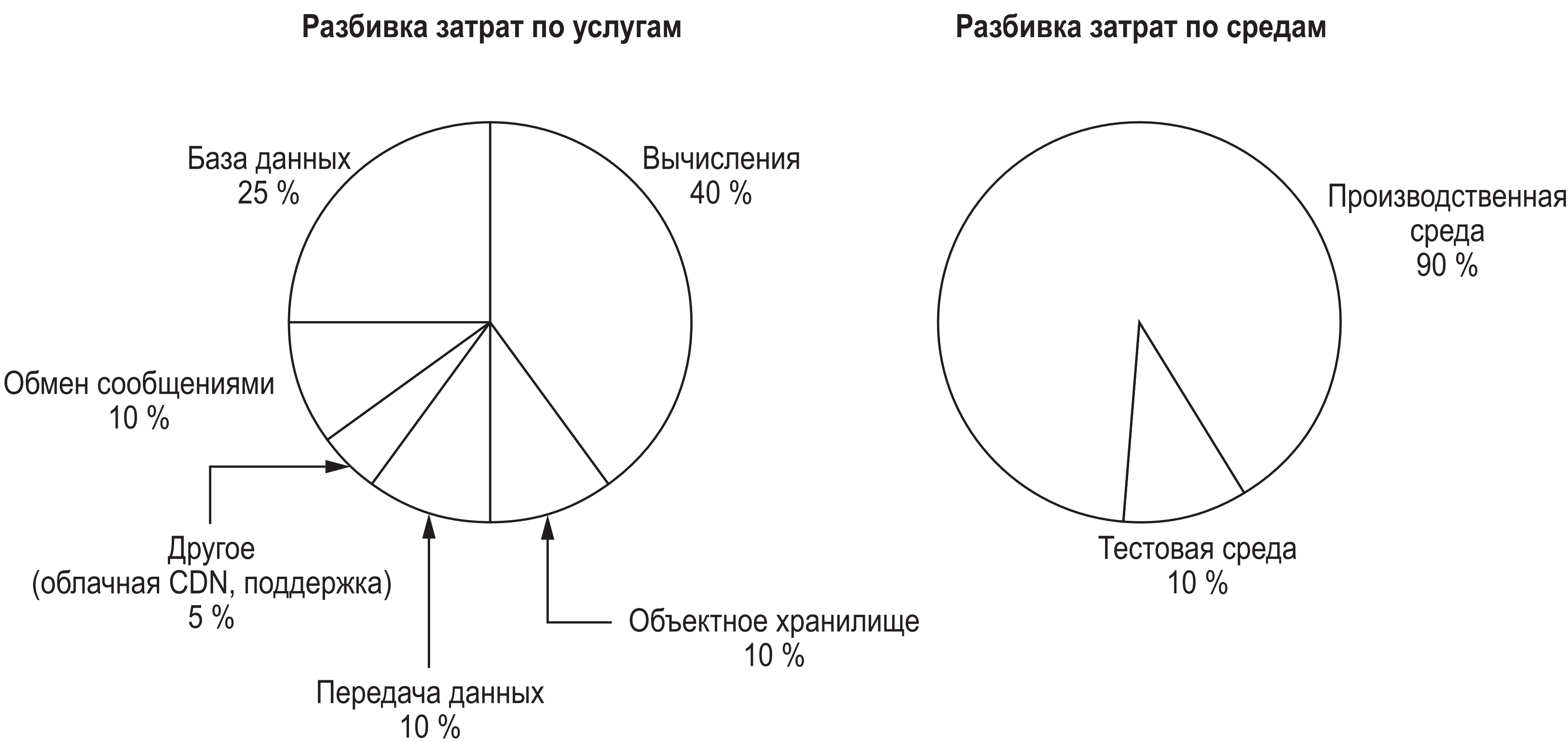


Рис. 12.1 Разбивка стоимости ресурсов по сервисам и средам с помощью тегов

Значительная часть расходов связана с вычислительными ресурсами, в частности с серверами. Если членам вашей команды необходимо поддерживать еще более крупную конференцию, им следует внимательно следить за ресурсами, которые они создают, и оптимизировать размер ресурсов в зависимости от их использования. Вы принимаете решение изучить методы управления размером и типами серверов, которые использует команда.

12.1.1 Способы тестирования для контроля затрат

Вы проверяете показатели конференции и использование ресурсов для каждого сервера. Ни один из серверов не вышел за рамки своего виртуального ЦП (vCPU) или памяти. По вашей грубой оценке, компании нужно не более 32 виртуальных ЦП для производственной среды. Команда инфраструктуры вашего клиента подтверждает, что максимальное использование не превышает 32 виртуальных ЦП.

ПРИМЕЧАНИЕ GCP использует термин *machine type* (тип машины) для обозначения предварительно заданной формы виртуальной машины с определенным соотношением виртуальных ЦП и памяти, соответствующим требованиям вашей рабочей нагрузки. Аналогично AWS использует термин *instance type* (тип экземпляра), а Azure – термин *size* (размер).

Однако общедоступное облако позволяет легко настроить сервер для использования 48 виртуальных ЦП. Ваш счет увеличивается на 50 % из-за дополнительного процессора, а вы даже не используете его полностью! Чтобы более активно контролировать затраты с помощью IaC, вы объединяете модульное тестирование из главы 6 и соблюдение политик из главы 8.

На рис. 12.2 показано добавление нового теста политики в конвейер поставки сервера. Тест проверяет каждый сервер, определенный в IaC, на предмет количества виртуальных ЦП и сравнивает его со значением, возвращаемым API GCP. Если информация API пре-

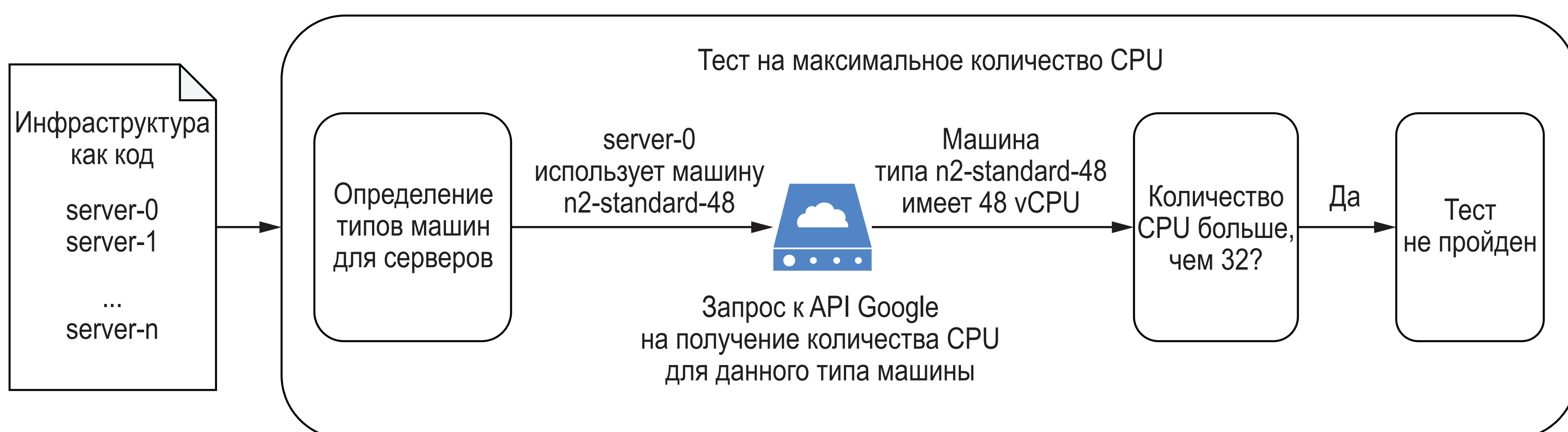


Рис. 12.2 Ваш тест должен проанализировать типы компьютеров из конфигураций серверов, проверить GCP API на количество процессоров и убедиться, что они не превышают лимит

вышает максимальный предел виртуальных ЦП, равный 32, тест не пройден.

Зачем вызывать API провайдера инфраструктуры для получения информации о виртуальных ЦП? Многие провайдеры предлагают API или клиентскую библиотеку для получения информации о каждом типе машины из своих каталогов. Вы можете использовать их для динамического получения информации о количестве процессоров и памяти.

Провайдеры инфраструктуры часто меняют технические параметры услуг. Вы не можете предугадать все возможные типы серверов на будущее. Разработка теста с вызовом API провайдера инфраструктуры для получения актуальной информации способствует соблюдению принципа развиваемости.

В листинге 12.1 показана реализация теста политики для проверки максимального предела виртуальных ЦП. Сначала мы создаем метод вызова API GCP для определения количества виртуальных ЦП для данного типа машины.

Листинг 12.1 Получение количества виртуальных ЦП для типа машины из GCP API

```
import googleapiclient.discovery

class MachineType():
    def __init__(self, gcp_json):
        self.name = gcp_json['name']
        self.cpus = gcp_json['guestCpus']
        self.ram = self._convert_mb_to_gb(
            gcp_json['memoryMb'])
        self.maxPersistentDisks = gcp_json[
            'maximumPersistentDisks']
        self.maxPersistentDiskSizeGb = gcp_json[
            'maximumPersistentDisksSizeGb']
        self.isSharedCpu = gcp_json['isSharedCpu']

    def _convert_mb_to_gb(self, mb):
        GIGABYTE = 1.0/1024
        return GIGABYTE * mb

def get_machine_type(project, zone, type):
    service = googleapiclient.discovery.build(
        'compute', 'v1')
    result = service.machineTypes().list(
        project=project,
        zone=zone,
        filter=f'name:"{type}"').execute()
    types = result['items'] if 'items' in result else None
    if len(types) != 1:
        return None
    return MachineType(types[0])
```


- ❶ Определяем объект типа машины для хранения любых атрибутов, которые вам может потребоваться проверить, включая количество виртуальных ЦП.
- ❷ Преобразовываем мегабайты памяти в гигабайты для единообразия единиц измерения.
- ❸ Вызываем API GCP для получения количества виртуальных ЦП для данного типа машины.
- ❹ Возвращаем объект типа машины с атрибутами виртуального ЦП и диска.

Эквиваленты AWS и Azure

Вы можете использовать Python SDK AWS для получения экземпляров EC2 и анализа типа экземпляра. Затем опишите типы экземпляров, чтобы получить информацию о виртуальном ЦП и памяти (<http://mng.bz/qYYK>).

Чтобы получить тип машины и единицы хранения (SKU) из Azure, используйте библиотеку Azure для Python. После получения размера из списка экземпляров вы можете вызвать API экземпляров ресурсов для получения информации о количестве процессоров и памяти (<http://mng.bz/YG1N>).

Теперь, когда вы выбрали новый тип машины, можно использовать готовую функцию для получения виртуальных ЦП и памяти. Затем вы пишете тест для анализа каждого сервера, определенного в вашей конфигурации, на предмет его типа. В листинге 12.2 мы получаем количество виртуальных ЦП для типа машины для списка серверов и проверяем, что количество виртуальных ЦП не превышает предел в 32.

Листинг 12.2 Тест политики для проверки того, что ресурсы серверов не превышают 32 виртуальных ЦП

```
import pytest
import os
import compute
import json

ENVIRONMENTS = ['testing', 'prod']
CONFIGURATION_FILE = 'main.tf.json'

PROJECT = os.environ['CLOUDSDK_CORE_PROJECT']

@pytest.fixture(scope="module")
def configuration():
    merged = []
    for environment in ENVIRONMENTS:
        with open(f'{environment}/{CONFIGURATION_FILE}', 'r') as f:
            environment_configuration = json.load(f)
            merged += environment_configuration['resource']
    return merged

def resources(configuration, resource_type):
```



```

resource_list = []
for resource in configuration:
    if resource_type in resource.keys():
        resource_name = list(
            resource[resource_type].keys())[0]
        resource_list.append(
            resource[resource_type]
            [resource_name])
return resource_list

@pytest.fixture
def servers(configuration):
    return resources(configuration,
                      'google_compute_instance')

def test_cpu_size_less_than_or_equal_to_limit(servers):
    CPU_LIMIT = 32
    non_compliant_servers = []
    for server in servers:
        type = compute.get_machine_type(
            PROJECT, server['zone'],
            server['machine_type'])
        if type.cpus > CPU_LIMIT:
            non_compliant_servers.append(server['name'])
    assert len(non_compliant_servers) == 0, \
        f'Servers found using over {CPU_LIMIT}' + \
        f' vCPUs: {non_compliant_servers}'

```

- ❶ Анализируем и извлекаем все конфигурации сервера JSON в тестовых и производственных средах.
- ❷ Устанавливаем ограничение ЦП на 32 – максимальное значение, необходимое для приложения.
- ❸ Инициализируем список некорректных серверов, количество виртуальных ЦП которых превышает лимит в 32.
- ❹ Для каждой конфигурации сервера извлекаем атрибут типа компьютера и вызываем API GCP для получения дополнительной информации.
- ❺ Если конфигурация сервера содержит тип с более чем 32 виртуальных ЦП, добавьте его в список некорректных серверов.
- ❻ Проверяем, что все серверы соответствуют лимиту ЦП. Если нет, возвращаем ошибку для серверов с числом процессоров более 32.

Вы настраиваете тест с мягким соблюдением политики. Мягкое соблюдение означает, что в принципе вы можете создать более дорогой ресурс, но сначала ваша команда должна проверить и подтвердить потребность в таком ресурсе. Вы можете изменить тип машины на более ресурсоемкий, если у вас есть экономическое обоснование.

Помимо проверки типов компьютеров на соответствие ограничениям виртуального ЦП и памяти, вам также может потребоваться добавить переопределения для уникальных архитектур или типов компьютеров, которые применимы к определенным случаям использования, например машинному обучению. Однако они стоят дороже, чем ресурсы общего назначения.

Убедитесь, что IaC по умолчанию использует ресурсы общего назначения. Машины или типы ресурсов общего назначения обычно обходятся дешевле. Если кому-то нужен специализированный и более дорогой ресурс, вы можете добавить его через процедуру утверждения расходов.

Другие тесты могут проводить проверку конкретных конфигураций, таких как запланированные перезагрузки, автоматическое масштабирование или частная сеть. Каждая из этих конфигураций способствует оптимизации стоимости ваших ресурсов. Представление ресурсов в форме сценариев конфигурации IaC позволяет вам убедиться, что конфигурация соответствует современным требованиям, что позволяет снизить затраты на ранних этапах процесса разработки.

12.1.2 Автоматизация оценки затрат

До этого момента тесты политики для контроля затрат фактически были вашей реакцией на уже состоявшиеся чрезмерные расходы. Что, если вам нужен упреждающий способ проверить, как изменятся расходы при изменении факторов затрат? Представьте, что вас интересует, как перенос вашего производственного сервера на машину типа `n2d-standard-16` (16 виртуальных ЦП) может повлиять на будущую стоимость другой трехчасовой конференции.

На рис. 12.3 показан рабочий процесс предварительной оценки стоимости пяти серверов с типом машины `n2d-standard-16`. После расчета цены вы можете добавить проверку политики, чтобы убедиться, что общая сумма не превышает ваш ежемесячный бюджет.

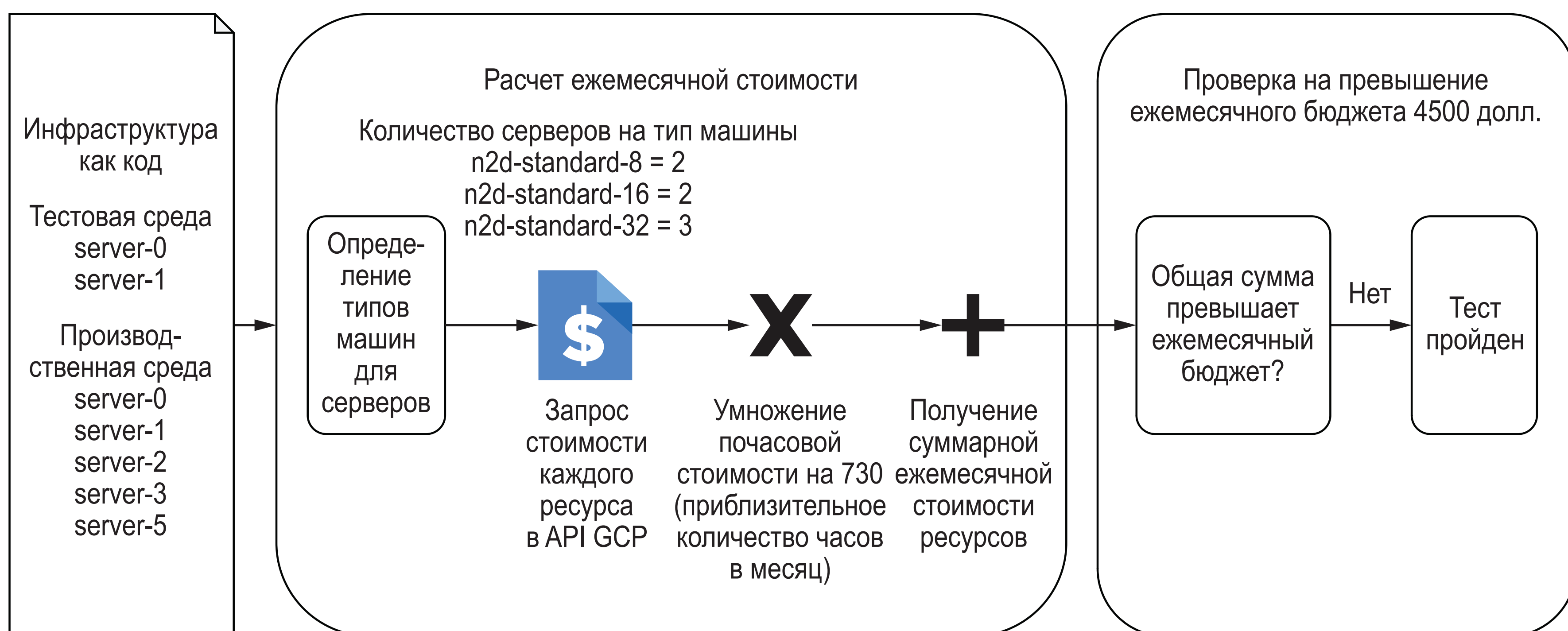


Рис. 12.3 Процесс оценки затрат извлекает из IaC типы машин, вычисляет ежемесячную стоимость ресурса и генерирует значение для сравнения с ожидаемым бюджетом

Процедура оценки затрат заключается в анализе атрибутов ресурсов IaC и вычислении их стоимости. Используйте оценку затрат,

чтобы убедиться, что ваши изменения укладываются в бюджет, или спланировать корректировки факторов затрат.

ОПРЕДЕЛЕНИЕ Оценка затрат (cost estimation) заключается в извлечении атрибутов ресурсов инфраструктуры и вычислении их общей стоимости.

Как оценка затрат помогает вам развивать вашу инфраструктуру? Оценка затрат обеспечивает дополнительную прозрачность факторов затрат, которые могут повлиять на вашу архитектуру. При изменении системы вы можете использовать эти тесты для составления бюджета и информирования команд о предполагаемом размере начислений за ресурсы.

Пример и инструменты оценки стоимости

Я написала минимальный код, чтобы продемонстрировать общий процесс оценки затрат. Для наглядности в тексте я опустила часть кода. Вы можете найти полный код по адресу <https://github.com/joatmon08/manning-book/tree/main/ch12>.

В примере используется API биллинга Google Cloud, который предлагает каталог услуг с ценами. Я также использую специализированную клиентскую библиотеку Python для облачного каталога для доступа к API биллинга (<http://mng.bz/mOOn>). В примере не учитываются специальные цены, такие как скидки на постоянное использование или вытесняемые инстансы (эквивалент спотовых инстансов в AWS).

Вы найдете несколько инструментов, которые выполняют оценку затрат. Каждый провайдер облачных услуг предлагает собственный пользовательский интерфейс оценщика затрат для ваших ресурсов. Другие инструменты реализуют более масштабируемый рабочий процесс, чем код моего примера, для анализа конфигурации, вызова API облачного провайдера и расчета сметы затрат. Я не буду перечислять их в этой главе, поскольку они часто меняются и зависят от провайдера облачных услуг и вашего инструментария IaC.

Получение стоимости единицы ресурса

Я рекомендую динамически запрашивать информацию из API каталога услуг облачного провайдера. Цена за единицу ресурса может меняться, и жесткое кодирование цен часто приводит к неправильной оценке затрат. Чтобы реализовать оценку затрат в этом примере, вам понадобится код для вызова каталога поставщика облачных услуг и получения цены за единицу в зависимости от типа вашей машины.

API каталога биллинга Google Cloud предлагает список услуг и SKU на основе цены за единицу ЦП или памяти (ОЗУ). В листинге 12.3 мы

получаем идентификатор службы Google Compute Engine. API каталога Google Cloud классифицирует цены на основе идентификаторов услуг, которые необходимо получать динамически.

Листинг 12.3 Получение стоимости услуги Google Compute Engine из каталога

```
from google.cloud import billing_v1

class ComputeService:
    def __init__(self):
        self.billing = \
            billing_v1.services.cloud_catalog.CloudCatalogClient()
        for result in self.billing.list_services():
            if result.display_name == 'Compute Engine':
                self.name = result.name
```

- ❶ Создаем клиента для API Google Cloud Billing Catalog с помощью библиотеки Python.
- ❷ Получаем идентификатор сервиса для Google Compute Engine в каталоге.

Эквиваленты AWS и Azure

Обновите клиентскую библиотеку GCP, чтобы она вызывала API AWS Cost Explorer (<http://mng.bz/5Qm4>) для AWS. В свою очередь, Azure предлагает открытую конечную точку REST API для розничных цен (<http://mng.bz/6X9G>). Вы можете написать дополнительный код для запроса информации из каталога.

Вы можете вызвать API Google Cloud Billing Catalog во второй раз, чтобы узнать цену типа машины, как показано в листинге 12.4. Используя идентификатор службы из предыдущего шага, мы получаем список SKU для службы Google Compute Engine. Далее наш код анализирует список SKU по типу и назначению машины и получает цену за единицу процессора или гигабайта памяти.

Листинг 12.4 Получение цены на ЦП и ОЗУ для SKU Google Compute Engine

```
from google.cloud import billing_v1

class ComputeSKU:
    def __init__(self, machine_type, service_name):
        self.billing = \
            billing_v1.services.cloud_catalog.CloudCatalogClient()
        self.service_name = service_name
        type_name = machine_type.split('-')
        self.family = type_name[0]
        self.exclude = [
            'custom',
            'preemptible',
```



```

        'sole tenancy',
        'commitment'
    ] if type_name[1] == 'standard' else []

def _filter(self, description):
    return not any(
        type in description for type in self.exclude
    )

def _get_unit_price(self, result):
    expression = result.pricing_info[0]
    unit_price = expression. \
        pricing_expression.tiered_rates[0].unit_price.nanos \
        if expression else 0
    category = result.category.resource_group
    if category == 'CPU':
        self.cpu_pricing = unit_price
    if category == 'RAM':
        self.ram_pricing = unit_price

def get_pricing(self, region):
    for result in self.billing.list_skus(parent=self.service_name):
        description = result.description.lower()
        if region in result.service_regions and \
            self.family in description and \
            self._filter(description):
            self._get_unit_price(result)
    return self.cpu_pricing, self.ram_pricing

```

- ❶ Создаем клиента с библиотекой Python для API каталога Google Cloud Billing.
- ❷ Для типа машины, например n2d-standard-16, извлекаем семейство машины (N2D) и назначение (стандарт) для определения SKU.
- ❸ Если вы используете компьютер стандартного типа, не ищите в каталоге какие-либо специализированные SKU Compute Service.
- ❹ Получаем цену за единицу ЦП или ОЗУ в нанодолларах (10^{-9}).
- ❺ Вызываем каталог биллинга Google Cloud и получаем список SKU для службы вычислений Compute Service.
- ❻ Находим SKU, соответствующие региону, семейству компьютеров и назначению типа машины, на основе его описания.

Каталог биллинга Google Cloud устанавливает цену за единицу ресурса в зависимости от количества процессоров и гигабайтов памяти. Поэтому поиск по названию типа машины невозможен. Вместо этого вам необходимо соотнести тип машины общего назначения с описанием в каталоге.

Расчет ежемесячной стоимости одного ресурса

Получив цену за единицу ЦП и ОЗУ для данного типа машины, вы можете использовать ее для расчета ежемесячных затрат на один экземпляр машины. В некоторых облачных каталогах цена за единицу устанавливается с коэффициентом. Например, в GCP используются

наноединицы, а это значит, что вам нужно еще и умножить результат на коэффициент. В листинге 12.5 показан код для расчета ежемесячной стоимости одного сервера. Мы умножаем цену за единицу на среднее количество часов в месяц (730) и на 10^{-9} .

Листинг 12.5 Расчет ежемесячной стоимости одного сервера

```
HOURS_IN_MONTH = 730 1
NANO_UNITS = 10**-9 1

def calculate_monthly_compute(machine_type, region):
    service_name = ComputeService().name 2
    sku = ComputeSKU(machine_type.name, service_name) 3
    cpu_price, ram_price = sku.get_pricing(region) 3

    cpu_cost = machine_type.cpus * cpu_price * \ 4
        HOURS_IN_MONTH if cpu_price else 0 4
    ram_cost = machine_type.ram * ram_price * \ 5
        HOURS_IN_MONTH if ram_price else 0 5
    return (cpu_cost + ram_cost) * NANO_UNITS 6
```

- 1 Задаем константу 730 часов в месяц и конвертируем нанодоллары в доллары (10^{-9}).
- 2 Получаем идентификатор службы Compute Engine из API каталога биллинга Google Cloud.
- 3 Задаем SKU для типа машины и получаем цену за единицу ЦП и ОЗУ.
- 4 Умножаем количество процессоров типа машины на цену за единицу и количество часов в месяце.
- 5 Умножаем количество гигабайтов памяти (ОЗУ) типа машины на цену за единицу и количество часов в месяце.
- 6 Суммируем стоимость ЦП и ОЗУ и конвертируем ее в доллары.

Теперь у вас есть минимальная форма расчета стоимости, позволяющая найти стоимость одного сервера. Благодаря первоначальному расчету стоимости для одного сервера вы можете проанализировать IaC для всех серверов, получить типы и регионы их компьютеров и рассчитать общую стоимость. В будущем вы сможете добавить дополнительную логику для получения номеров SKU для других служб, таких как базы данных или обмен сообщениями.

Проверка соответствия ожидаемых затрат бюджету

Вы понимаете, что можете пойти дальше в процессе расчета затрат. Вы пишете тест с мягкими условиями соблюдения политики, чтобы проверить, не превышают ли предполагаемые затраты ваш ежемесячный бюджет. Например, ваш клиент сообщает вам, что ежемесячный бюджет конференции не должен превышать 4500 долл. США. Вы можете сравнить свою оценку затрат с бюджетом и заблаговременно выявить любые факторы, способствующие превышению бюджета.

Вам нужен тест для вычисления новой стоимости серверов и сравнения ее с бюджетом. В листинге 12.6 показан анализ IaC для всех

серверов и подсчет количества серверов с определенными типами компьютеров и регионами.

Листинг 12.6 Анализ IaC для всех серверов

```

from compute import get_machine_type
import pytest
import os
import json

ENVIRONMENTS = ['testing', 'prod']
CONFIGURATION_FILE = 'main.tf.json'

@pytest.fixture(scope="module")
def configuration():
    merged = []
    for environment in ENVIRONMENTS:
        with open(f'{environment}/{CONFIGURATION_FILE}', 'r') as f:
            environment_configuration = json.load(f)
            merged += environment_configuration['resource']
    return merged

@pytest.fixture
def servers(configuration):
    servers = dict()
    server_configs = resources(configuration,
                                'google_compute_instance')
    for server in server_configs:
        region = server['zone'].rsplit('-', 1)[0]
        machine_type = server['machine_type']
        key = f'{region},{machine_type}'
        if key not in servers:
            type = get_machine_type(
                PROJECT, server['zone'],
                machine_type)
            servers[key] = {
                'type': type,
                'num_servers': 1
            }
        else:
            servers[key]['num_servers'] += 1
    return servers

```

- ❶ Считываем каждый файл конфигурации, определяющий каждую среду, например тестовую и производственную.
- ❷ Для каждого сервера в файле конфигурации создаем список их регионов и типов компьютеров.
- ❸ Вызываем Google Compute API и получаем сведения о типе машины, например количество процессоров и памяти.
- ❹ Отслеживаем количество серверов с определенным типом машины и регионом, чтобы упростить поиск SKU, который вам необходимо получить.

Вы можете вызвать эти методы в своем тесте, чтобы получить информацию о стоимости каждого типа машины в определенном регионе и суммировать общую стоимость. Тест в листинге 12.7 проверяет, превышает ли общая стоимость ежемесячный бюджет в 4500 долл. США.

Листинг 12.7 Получение цены на ЦП и ОЗУ для заданных SKU

```
from estimation import calculate_monthly_compute

ENVIRONMENTS = ['testing', 'prod']
CONFIGURATION_FILE = 'main.tf.json'

PROJECT = os.environ['CLOUDSDK_CORE_PROJECT']
MONTHTLY_COMPUTE_BUDGET = 4500 ❶

def test_monthly_compute_budget_not_exceeded(servers): ❷
    total = 0
    for key, value in servers.items():
        region, _ = key.split(',')
        total += calculate_monthly_compute(value['type'], region) * \ ❸
            value['num_servers'] ❸
    assert total < MONTHTLY_COMPUTE_BUDGET ❹
```

- ❶ Задаем константу, хранящую ожидаемый ежемесячный бюджет на вычисления.
- ❷ Проверяем, что стоимость наших серверов не превышает ежемесячный бюджет вычислений.
- ❸ Рассчитываем ежемесячные затраты на сервер на основе типа машины и региона, умноженные на количество серверов для этого типа машины, и находим общую сумму.
- ❹ Проверяем, не превышает ли предполагаемая общая стоимость ежемесячный бюджет вычислений.

Теперь у вас есть тест, позволяющий найти общую ежемесячную стоимость вычислительных ресурсов и сравнить ее со своим бюджетом! Каждый раз, когда кто-то меняет инфраструктуру, тест вычисляет новую стоимость системы.

Оценка затрат дает вам общее представление о стоимости вашей инфраструктуры, но не обязательно совпадает с суммой в счете от вашего провайдера. Вам придется учитывать определенную погрешность. Если ваша оценка превышает ежемесячный бюджет, это может означать, что вам необходимо пересмотреть размер и использование ресурсов. Вы также со временем сможете скорректировать свой ежемесячный бюджет в зависимости от роста инфраструктуры.

Непрерывная поставка с оценкой стоимости

Как убедиться, что изменения инфраструктуры не ведут к превышению бюджета? Каждый раз, когда вы меняете сценарий IaC и отправляете его в репозиторий, вы оцениваете затраты и проверяете

соответствие бюджету. Тесты на соблюдение бюджета в вашем конвейере помогут вам вовремя выявить дорогостоящие изменения инфраструктуры и усовершенствовать ресурс в среде тестирования. Этот процесс предотвращает незапланированные расходы в производственной среде.

Предположим, что вы хотите добавить еще один сервер в тестовую среду для другой конференции. На рис. 12.4 показано добавление в конфигурацию еще одного сервера с типом машины `n2d-standard-8`. Конвейер запускает тест для расчета ежемесячных затрат на новый сервер и сверяет их с ежемесячным бюджетом.

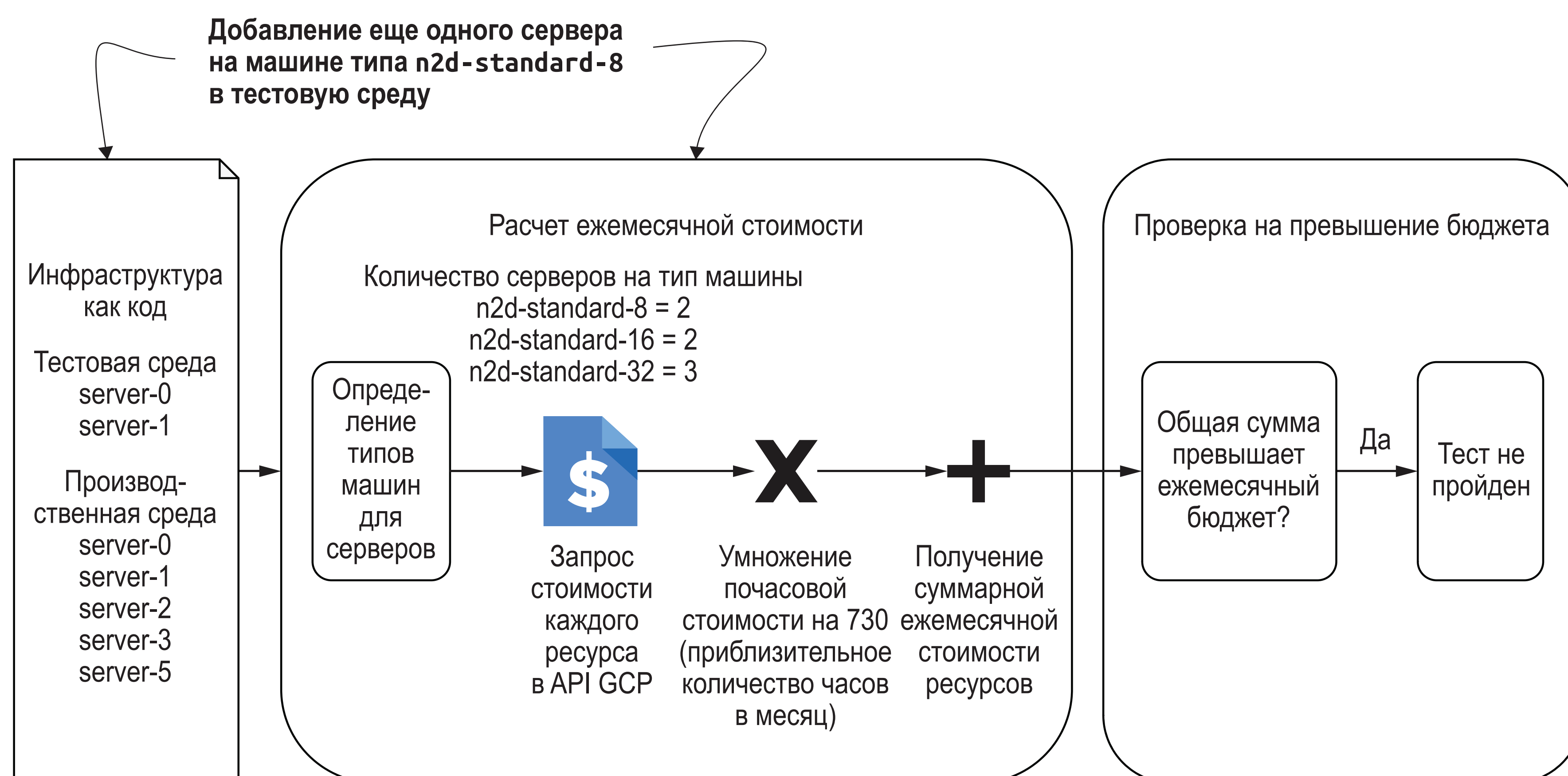


Рис. 12.4 Добавление еще одного сервера приводит к превышению бюджета в 4500 долл., поэтому тест не пройден

Вы отправляете конфигурацию в репозиторий, и ваш конвейер поставки запускает тесты для проверки соответствия бюджету. Конвейер остановился! Вы проверяете логи и обнаруживаете, что расчетная величина затрат превышает ожидаемый ежемесячный бюджет:

```
$ pytest test_budget.py
FAILED test_budget.py::test_monthly_compute_budget_not_exceeded -
➔assert 4687.6161600000005 < 4500
```

Вы консультируетесь со своей финансовой командой. Финансовый аналитик согласен с вашим аргументом, что ради размещения нового тестового экземпляра можно немного увеличить бюджет. Вы обновляете граничное значение ежемесячного бюджета до 4700 долларов.

Независимо от того, разрабатываете ли вы механизм оценки затрат самостоятельно или используете какой-либо инструмент, я рекомендую добавить его в свои конвейеры поставки в качестве еще

одной проверки политики. Оценка затрат помогает выбрать параметры и способ применения ресурса. Не следует рассматривать бюджетные ограничения только как причину отмены изменений до отправки в производство. Выход за рамки бюджета – хороший повод более тщательно проанализировать свою потребность в ресурсе.

Не пытайтесь учесть каждый фактор при оценке затрат. Ограничьтесь ресурсами, которые формируют основную часть ваших затрат. В нашем примере основное внимание уделяется вычислительным ресурсам, таким как серверы, которые часто вносят значительный вклад в стоимость. В вашем случае это могут быть другие ресурсы, такие как базы данных или системы обмена сообщениями.

Всегда подвергайте сомнению точность вашей оценки затрат! Вы не можете точно предсказать, какие ресурсы вы создадите или как вы их используете. Например, вам может быть сложно оценить стоимость передачи данных между регионами или службами, пока не произойдет обмен трафиком. Сопоставьте свою смету расходов с реальным ежемесячным счетом и постарайтесь понять, какие факторы затрат формируют разницу.

Ежемесячное сравнение оценок затрат с реальными счетами помогает выявить изменения и составить бюджет фактических затрат путем умножения оценки на поправочный коэффициент. В оставшейся части главы мы обсудим другие способы сокращения потерь в облаке и оптимизации затрат, кроме оценки и тестирования.

Упражнение 12.1

Какие из приведенных ниже утверждений верны для данного кода (выберите все подходящие варианты):

```
HOURS_IN_MONTH = 730
MONTHLY_BUDGET = 5000
DATABASE_COST_PER_HOUR = 5
NUM_DATABASES = 2
BUFFER = 0.1
```

```
def test_monthly_budget_not_exceeded():
    total = HOURS_IN_MONTH * NUM_DATABASES * DATABASE_COST_PER_HOUR
    assert total < MONTHLY_BUDGET + MONTHLY_BUDGET * BUFFER
```

- a** тест будет пройден, поскольку стоимость базы данных находится в пределах бюджета;
- b** тест оценивает ежемесячную стоимость баз данных;
- c** тест не учитывает различные типы баз данных;
- d** в ходе теста вычисляется ежемесячная стоимость одного экземпляра базы данных;
- e** тест включает в себя буфер в размере 10 % на случай превышения затрат в качестве мягкой политики.

Ответы на упражнения приведены в приложении В.

12.2 Сокращение объема облачных излишков

Вы можете использовать IaC для реализации упреждающих мер по управлению факторами затрат для облачных вычислений. Однако вам необходимо сочетать их с другими приемами, чтобы продолжать сокращать и оптимизировать затраты. В конце концов, вашему клиенту в недавнем примере совсем не нравится счет в 10 000 долларов за трехчасовую конференцию!

Если вы запрашиваете большой сервер, но не используете весь ЦП или память, вы напрасно оплачиваете излишек облачных ресурсов. Это хорошая возможность снизить затраты на облачные вычисления! Один из подходов, который вы можете использовать для снижения расходов, заключается в устранении ненужных, не используемых в данный момент или недостаточно используемых ресурсов инфраструктуры.

ОПРЕДЕЛЕНИЕ *Облачные излишки (cloud waste)* – это неиспользуемые или недостаточно используемые ресурсы облачной инфраструктуры.

Вы можете сократить облачные излишки, удалив или остановив неиспользуемые ресурсы, применив планирование или масштабирование экземпляров в зависимости от использования, заранее оценив правильные параметры или тип ресурсов для своей системы. Эти подходы схематически показаны на рис. 12.5.



Рис. 12.5 Вы можете сократить облачные излишки, удалив неиспользуемые ресурсы или запланировав и определив размеры ресурсов в соответствии с паттернами использования

Выявление облачных излишков часто представляет собой очевидную реакцию на неожиданно большой счет за облачные услуги. Однако вы можете использовать эти методы также в своем центре обработки данных, особенно в частном облаке. Хотя они не дают немедленной краткосрочной выгоды, они помогают оптимизировать использование ресурсов центра обработки данных и снизить затраты в долгосрочной перспективе.

12.2.1 *Остановка непомеченных или неиспользуемых ресурсов*

Иногда вы и ваша команда будете создавать ресурсы инфраструктуры для тестирования или другой конфигурации. Рано или поздно вы забудете о каких-то ресурсах, пока они не появятся в вашем счете за облачные услуги. В качестве первого шага по сокращению облачных излишков вы можете выявить неиспользуемые ресурсы и удалить их.

Помните, что ваша задача – снизить стоимость проведения конференции для вашего клиента. Можете ли вы сократить расходы за счет выявления неиспользуемых ресурсов? Да! Иногда команды создают ресурсы для тестирования и забывают их удалить.

Представьте, что вы получили список серверов в своих проектах Google Cloud (табл. 12.2) и проанализировали его. Хотя в основном все серверы в тестовой и производственной средах имеют теги, вы обнаружили два экземпляра без тегов. Машины n2d-standard-16 обходятся компании примерно в 700 долл. (7 % от общего ежемесячного счета).

Таблица 12.2 Стоимость серверов в зависимости от типа и среды

Тип машины	Среда	Кол-во серверов	Подытог, долл.
n2d-standard-8	Тестирование	2	400
n2d-standard-16	Производство	2	700
n2d-standard-32	Производство	3	2900
Итог, долл.			4000

Вы спрашиваете членов команды о непомеченных экземплярах в производственной среде. Оказывается, они создали серверы для проверки приложения в «песочнице», но никогда их не использовали. Просто для подстраховки вы проверяете показатели использования сервера за месяц, и они действительно равны нулю. Вы обнаружили облачные излишки!

Команда создала серверы с помощью IaC. Вы вносите изменения в конфигурацию, чтобы удалить неиспользуемые экземпляры. При удалении серверов удаляются также диски и ресурсы, подключенные к экземплярам. К счастью, счет за облако для следующей конференции отражает это сокращение.

Зачем вам проверять использование сервера по метрикам и опросам членов команды? Вы же не хотите случайно удалить нужный ресурс! Иногда в инфраструктуре существуют неожиданные зависимости от того, что кажется неиспользуемым ресурсом.

Убедитесь, что все ресурсы, которые вы собираетесь удалить, не имеют дополнительных зависимостей. Если у вас есть опасения по поводу удаления непомеченного или неиспользуемого ресурса, вы всег-

да можете остановить ресурс на неделю или две, подождать, чтобы определить, не нарушает ли он работу системы, а затем удалить его.

12.2.2 Запуск и остановка ресурсов по расписанию

Ваш следующий счет за облачные услуги будет на 7 % меньше благодаря удалению неиспользуемых серверов. Однако финансовый отдел хочет, чтобы вы сократили его еще больше. Вы ломаете голову над этим, пока не поговорите с одним из членов команды клиента. Он сообщает, что команда никогда не проводит тестирование и не использует ресурсы своей инфраструктуры по выходным. Клиенту нужен доступ к платформе по выходным только непосредственно перед конференцией.

Не могли бы вы найти способ выключать серверы в пятницу вечером и включать их в понедельник утром? В таком случае провайдер не будет взимать плату за 48 часов отключения серверов. Планирование регулярного отключения означает снижение затрат.

Вы узнали, что GCP определяет график выключения экземпляра с помощью политики вычислительных ресурсов (<http://mng.bz/o25N>). Вы запускаете серверы каждый понедельник и выключаете их каждую субботу, как показано на рис. 12.6.

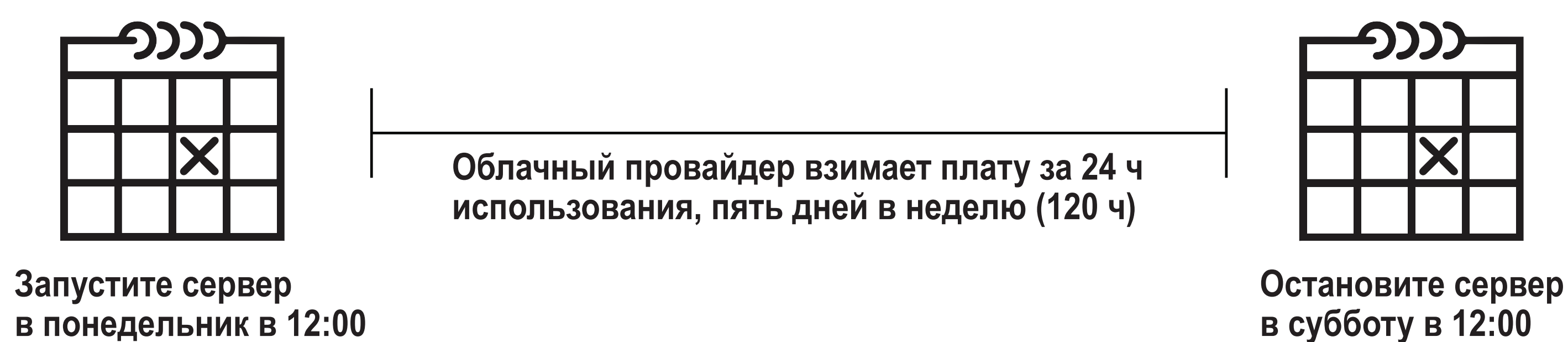


Рис. 12.6 Вы можете снизить затраты, запланировав запуск и остановку ресурса, когда он не используется

Выключение экземпляров по расписанию снижает затраты на эксплуатацию серверов. Однако этот метод работает только в том случае, если вы хорошо понимаете поведение своей системы в целом. Запуск и остановка ресурсов по расписанию может, например, помешать команде разработчиков.

Некоторые приложения не обладают отказоустойчивостью и останутся в состоянии сбоя даже в случае успешного перезапуска ресурса. Как правило, графики отключения ресурсов применяются только в тестовых средах. Отключение по расписанию дает вам возможность проверить устойчивость вашей системы, поскольку каждые выходные у вас фактически запланировано выпадение ресурсов.

В листинге 12.8 реализована ресурсная политика для планирования экземпляров в GCP. Срок действия расписания истекает за неделю до конференции, поэтому серверы не отключаются на выходных. Команде разработчиков, возможно, придется поработать над платформой за несколько дней до конференции.

Листинг 12.8 Создание ресурсной политики для планирования работы экземпляра

```
def build(name, region, week_before_conference):
    expiration_time = datetime.strptime(
        week_before_conference,
        '%Y-%m-%d').replace(
            tzinfo=timezone.utc).isoformat().replace(
                '+00:00', 'Z')
    return {
        'google_compute_resource_policy': {
            'weekend': {
                'name': name,
                'region': region,
                'description':
                    'start and stop instances over the weekend',
                'instance_schedule_policy': {
                    'vm_start_schedule': {
                        'schedule': '0 0 * * MON'
                    },
                    'vm_stop_schedule': {
                        'schedule': '0 0 * * SAT'
                    },
                    'time_zone': 'US/Central',
                    'expiration_time': expiration_time
                }
            }
        }
    }
```

- ❶ Срок действия расписания истекает за неделю до конференции, формат даты RFC 3339.
- ❷ Создаем политику вычислительных ресурсов с расписанием экземпляра.
- ❸ Запускаем виртуальные машины каждый понедельник в полночь.
- ❹ Останавливаем виртуальные машины каждую субботу в полночь.
- ❺ Привязываем расписание к центральному часовому поясу США, поскольку группы разработчиков работают в центральной части США.

Эквиваленты AWS и Azure

Другие поставщики общедоступных облаков обычно предлагают аналогичные возможности автоматизации для запуска и остановки виртуальных машин по расписанию. Например, AWS использует планировщик экземпляров для запуска и остановки серверов и баз данных (<http://mng.bz/nNev>). Аналогично Azure использует рабочий процесс запуска и остановки виртуальной машины на основе функций Azure (<http://mng.bz/v6Xx>).

Если ваша общедоступная или частная облачная платформа не поддерживает функционал запланированного отключения, вам придется написать свою систему автоматизации, которая будет работать по ре-

гулярному расписанию. Я делала это разными способами, включая бес-серверные функции, задания cron в оркестраторах контейнеров или запланированные запуски во фреймворках непрерывной интеграции.

Вместо того чтобы использовать семь серверов по 730 часов в месяц, вы используете их примерно на 144 часа меньше (при условии трех выходных в месяц и 48-часового отключения). Используя код оценки затрат, вы выполняете расчет для 586 часов в месяц. Выходит, что вы снизили общую стоимость на 700 долл. (7 % от общего ежемесячного счета)!

В примере расписание добавляется в тестовую среду. Однако вы можете добавить график перезагрузки в производственную среду, если она имеет циклические закономерности использования. Например, конференц-платформа потребляет ресурсы по требованию три часа и обслуживает пользовательский трафик только в течение рабочей недели. Отключение серверов и баз данных на 48 часов не нарушает пользовательский трафик. Но вряд ли стоит внедрять график перезагрузки в производственной среде, которая непрерывно обслуживает запросы.

12.2.3 *Выбор правильного типа и размера ресурса*

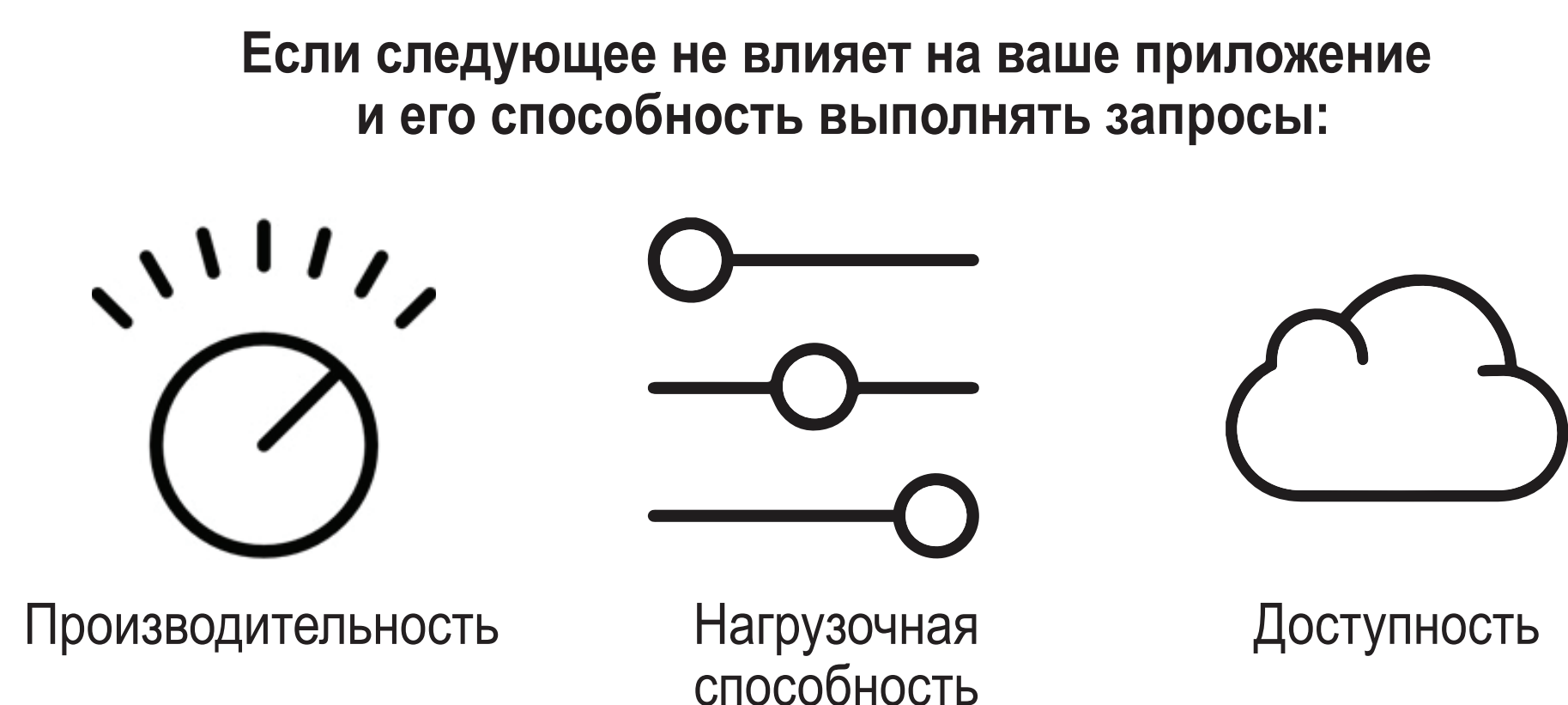
Если вашей производственной среде необходимо обслуживать клиентов 24 часа в сутки, семь дней в неделю, вы все равно можете сократить облачные излишки без расписания ресурсов, просто оценив их типы и размеры. Многие ресурсы не используют полностью свой процессор или память.

Часто мы выделяем большие ресурсы, потому что не знаем точно свои потребности. После эксплуатации системы в течение некоторого времени вы можете настроить размер ресурса в соответствии с его фактическим использованием. Вы можете снизить затраты, изменив тип ресурса, размер, резервирование, реплики или даже поставщика облачных услуг, как показано на рис. 12.7.

Вы решаете изучить конференц-платформу вашего клиента, чтобы найти в облаке излишки по типам и размерам ресурсов. Вы понимаете, что снизить затраты вычислительных ресурсов невозможно, поэтому проверяете базу данных (Cloud SQL). Команда предоставила рабочей базе данных твердотельный накопитель (SSD) емкостью 4 ТБ, как указано в табл. 12.3.

Таблица 12.3. Счета за облачные услуги по предложениям и ресурсам

Услуга	Тип	Среда	Кол-во	Подитог, долл.
Cloud SQL				2500
	db-standard-1, 400 GB SSD, 600 GB backup	Тестирование	1	250
	db-standard-4, 4 TB SSD, 6 TB backup	Производство	1	2250



Вы можете снизить стоимость инфраструктуры, изменив ресурс...

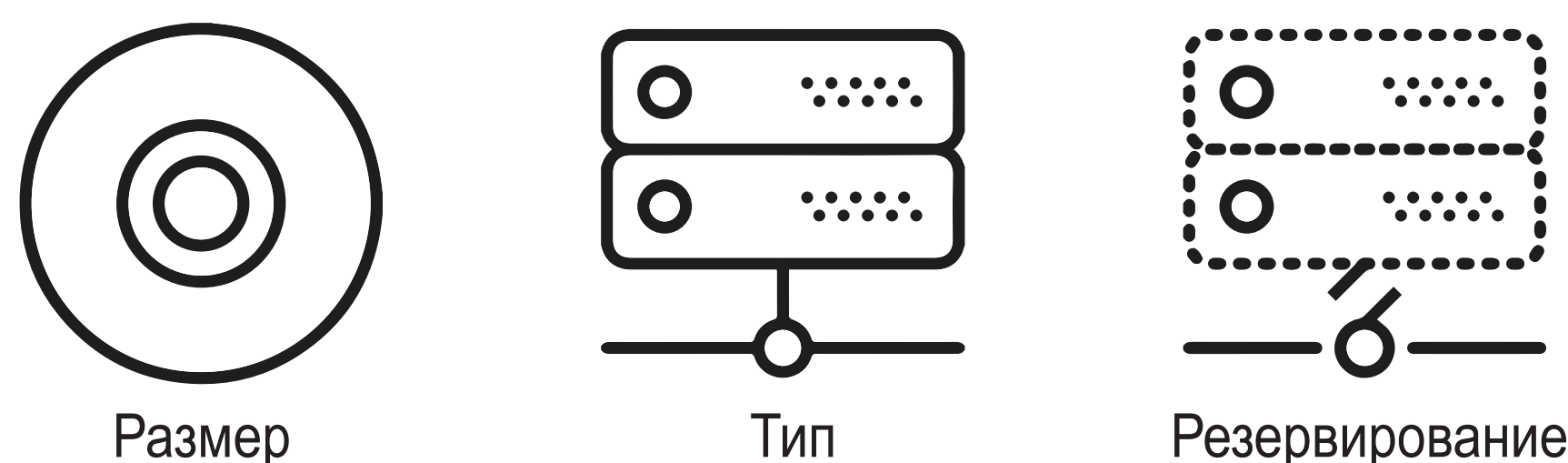


Рис. 12.7 Вы можете изменить атрибуты ресурса, чтобы полнее использовать его и снизить стоимость излишков

Проверив метрики и использование базы данных, вы понимаете, что для нее нужен только SSD на 1 ТБ. Вы обновляете размер диска базы данных в своей IaC. Исправление размера снижает стоимость базы данных на 1350 долл. (22,5 % от общего ежемесячного платежа)!

Излишки ресурсов могут возникнуть по разным причинам. Если ресурс использует дорогостоящий тип машины, имеет смысл задуматься о переходе на более дешевый экземпляр. Вам нужно спросить себя и свою команду: «Нужна ли нам эта высокопроизводительная база данных в тестовой среде, если мы не проводим тесты производительности?»

Возможно, нет! Выбор правильного размера и типа ресурса для конкретной среды может занять несколько итераций. Постарайтесь выбрать ресурсы, которые достаточно хорошо имитируют производство, не будучи эквивалентами по стоимости.

В примере с конференцией у вас могут быть три экземпляра `n2d-standard-32` в рабочей среде и три экземпляра `n2d-standard-8` в тестовой среде. Конфигурация по-прежнему тестирует три экземпляра приложения без затрат на 72 ЦП.

Иногда вы можете изменить тип выделения ресурса. GCP и многие другие провайдеры облачных услуг предлагают *спотовый* (spot), или *вытесняемый* (preemptible), тип ресурса. Стоимость такого ресурса меньше, но провайдер облачных услуг оставляет за собой право внезапно остановить ресурс и передать процессор или память другому клиенту. Хотя использование вытесняемых ресурсов может снизить затраты, хорошенько подумайте, смогут ли ваше приложение и система справиться с внезапным отключением ресурса.

12.2.4 Применение автомасштабирования

Вы постарались выявить как можно больше облачных излишков в своей среде, но при этом хотите еще больше сократить расходы. Для многих систем свойственно наличие характерных колебаний

клиентской нагрузки, поэтому они не нуждаются в непрерывном предоставлении максимально возможных ресурсов на всем протяжении работы.

Например, конференц-платформе потребовалось 100 % мощности всего на три часа конференции! Можно ли автоматически увеличивать или уменьшать количество серверов в зависимости от спроса?

На рис. 12.8 показан порог использования ЦП 75 %, в соответствии с которым группа управляемых экземпляров GCP запускает и останавливает серверы. Этот механизм увеличивает и уменьшает размер группы в зависимости от спроса.

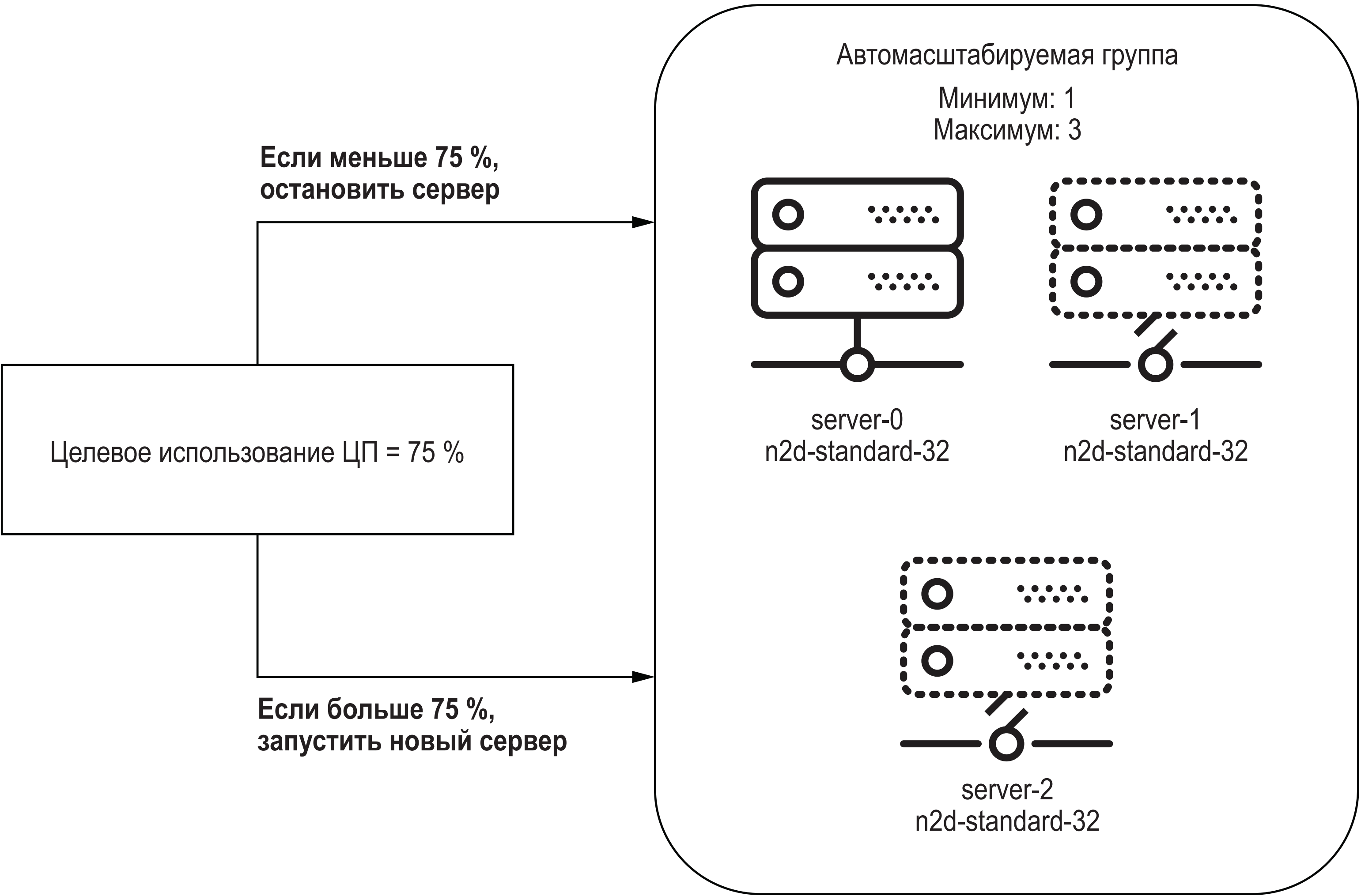


Рис. 12.8 Группа автомасштабирования отслеживает целевой коэффициент использования, на основании которого запускает и останавливает ресурсы для автоматической корректировки потребления

Допустим, вы добавили автомасштабирование для каждой группы серверов. Автомасштабирование увеличивает или уменьшает количество ресурсов в группе на основе таких показателей, как ЦП или память. Многие провайдеры общедоступных облаков предлагают групповой ресурс автоматического масштабирования, который можно создать с помощью IaC.

ОПРЕДЕЛЕНИЕ *Автомасштабирование (autoscaling)* – это практика автоматического увеличения или уменьшения количества ресурсов в группе на основе метрики.

Для автоматического масштабирования GCP необходимо установить пороговое значение, по достижении которого объем ресурсов

будет меняться в ту или иную сторону. Большую часть месяца с низким трафиком вы рассчитываете использовать только один сервер. Однако когда через вашу конференц-платформу проходит пиковый трафик, вам нужны максимум три сервера. Вы решаете использовать показатель загрузки ЦП и устанавливаете целевое значение 75 %, а затем обновляете IaC для серверов. В листинге 12.9 вы заменяете исходную политику планирования ресурсов серверов и экземпляров управляемой группой экземпляров и политикой автоматического масштабирования. Расписание автомасштабирования запускается каждое утро, увеличивает или уменьшает число экземпляров для достижения загрузки ЦП на 75 % и масштабирует экземпляры до нуля каждый вечер.

Листинг 12.9 Создание группы автомасштабирования на основе загрузки ЦП

```
def build(name, machine_type, zone,
        min, max, cpu_utilization,
        cooldown=60,
        network='default'):
    region = zone.rsplit('-', 1)[0]
    return [{
        'google_compute_autoscaler': {
            name: {
                'name': name,
                'zone': zone,
                'target': '${google_compute_instance_group_manager.' +
                f'{name}.id}}',
                'autoscaling_policy': {
                    'max_replicas': max,
                    'min_replicas': 0,
                    'cooldown_period': cooldown,
                    'cpu_utilization': {
                        'target': cpu_utilization
                    },
                    'scaling_schedules': {
                        'name': 'weekday-scaleup',
                        'min_required_replicas': min,
                        'schedule': '0 6 * * MON-FRI',
                        'duration_sec': '57600',
                        'time_zone': 'US/Central'
                    }
                }
            }
        }
    ]
```

- ① Прикрепляем группу экземпляров к ресурсу автомасштабирования. Группа экземпляров опущена для ясности.
- ② Устанавливаем максимальное количество реплик для масштабирования, если загрузка ЦП превышает 75 %.

- ③ По умолчанию устанавливаем минимальное количество реплик равным нулю, что означает остановку всех виртуальных машин.
- ④ Используем загрузку ЦП в качестве целевого показателя для группы автоматического масштабирования.
- ⑤ Устанавливаем график масштабирования, который увеличивает минимальное количество реплик каждое утро с понедельника по пятницу в соответствии с закономерностями рабочей нагрузки команды разработчиков.

Эквиваленты AWS и Azure

GCP присоединяет управляемые группы экземпляров к политикам автомасштабирования. GCP не позволяет прикрепить ресурсную политику. Вы должны реализовать расписание в группе автомасштабирования.

Другие провайдеры общедоступных облаков также предлагают возможности автоматического масштабирования серверов, а иногда и баз данных. AWS использует группы автомасштабирования. Azure использует правила автомасштабирования для масштабируемых наборов.

В этом примере вы можете задать график масштабирования, чтобы имитировать отключение на выходных, которое вы реализовали ранее. В целом необходимо придерживаться подхода с паттернами модулей для создания модуля автомасштабирования. Модуль должен установить целевую метрику по умолчанию в зависимости от вашей рабочей нагрузки.

Если у вас есть уникальная рабочая нагрузка, которая не соответствует предустановленным целевым метрикам модуля, можно начать с целевой нагрузки ЦП или памяти по умолчанию и скорректировать ее через какое-то время. При развертывании группы экземпляров примените методику blue-green развертывания из главы 9, чтобы заменить активные рабочие нагрузки или экземпляры. Добавление расписания и группы автомасштабирования не должно нарушать работу приложений.

Чтобы и другие команды захотели использовать группы автомасштабирования и планирование, создайте несколько тестов политики и покажите, что ваша группа автомасштабирования снижает облачные излишки. Например, тест может подтвердить, что ваш IaC не имеет независимых серверов и содержит только группы автоматического масштабирования. Тест побудит команду воспользоваться преимуществами эластичности.

Еще один тест, который имеет смысл добавить, проверяет максимальный лимит реплик. Предположим, ваше приложение потребляет неожиданно много ресурсов ЦП или памяти либо злоумышленник внедряет на ваш компьютер исполняемый файл для майнинга криптовалюты. Нужно сделать так, чтобы группа автомасштабирования не смогла автоматически увеличить свою емкость до 100 компьютеров.

12.2.5 Установка тега истечения срока действия ресурса

Вы можете сократить облачные излишки за счет динамического увеличения и уменьшения ресурсов в зависимости от их использования, но вам также необходимо что-то делать с ресурсами, создаваемыми вручную по требованию. Например, члены клиентской команды жалуются, что им часто приходится создавать серверы-песочницы для тестирования. Однако про серверы нередко забывают. Можно ли задать максимальный срок жизни серверов, которые никто не обновляет и не использует?

Вы решили обновить свой модуль тегов, чтобы прикрепить новый тег для даты истечения срока жизни ресурса в тестовой среде. Напомню, что для создания стандартных тегов вы можете использовать паттерн «прототип» (глава 3). После применения тестов политики для проверки соответствия тегов, описанных в главе 8, вы знаете, что у каждого ресурса в тестовой среде будет срок жизни.

Например, члены команды могут создать сервер с датой истечения срока жизни 2 февраля, как показано на рис. 12.9. Но вскоре они обновляют сервер. В рамках обновления модуль тегов получает текущую дату (5 февраля), добавляет семь дней и обновляет тег на сервере до новой даты (12 февраля).



Рис. 12.9 Добавление в модуль тегов даты истечения срока жизни ресурса спустя одну неделю после обновления

Зачем задавать срок жизни ресурса как часть модуля тегов? Ваш модуль тегов должен применяться ко всей IaC. Это позволяет вам задать *продолжительность по умолчанию* в семь дней и применить ее ко всем ресурсам инфраструктуры.

Вы также можете решать, когда применять тег срока жизни как часть модуля. Модуль применяет тег срока жизни только в том слу-

чае, если вы не создаете ресурс в производственной среде или не запускаете его постоянно в среде тестирования. В листинге 12.10 обновляется модуль прототипа для тегов по умолчанию с указанием даты истечения срока.

Листинг 12.10 Модуль тегов со сроком жизни ресурса

```
import datetime

EXPIRATION_DATE_FORMAT = '%Y-%m-%d' ❶
EXPIRATION_NUMBER_OF_DAYS = 7 ❷

class DefaultTags():
    def __init__(self, environment, long_term=False):
        self.tags = {
            'customer': 'community',
            'automated': True,
            'cost_center': 123456,
            'environment': environment
        }
        if environment != 'prod' and not long_term: ❸
            self._set_expiration() ❸

    def get(self):
        return self.tags

    def _set_expiration(self):
        expiration_date = ( ❹
            datetime.datetime.now() + ❹
            datetime.timedelta( ❹
                days=EXPIRATION_NUMBER_OF_DAYS) ❹
        ).strftime(EXPIRATION_DATE_FORMAT) ❺
        self.tags['expiration'] = expiration_date
```

- ❶ Задаем формат даты в виде строки «год – месяц – день».
- ❷ Вычисляем дату истечения срока жизни через семь дней от текущей даты.
- ❸ Устанавливаем тег срока жизни, если вы не создаете ресурс в производственной среде или не отмечаете его как долгосрочный ресурс.
- ❹ Вычисляем дату истечения срока жизни на семь дней от текущей даты.
- ❺ Задаем формат даты в виде строки «год – месяц – день».

По умолчанию вы устанавливаете одну неделю, поскольку она дает членам команды достаточно времени для разработки и тестирования ресурса. При необходимости они всегда могут продлить срок еще на неделю, запустив конвейер поставки для автоматического обновления тега. Но вам необходимо предусмотреть возможность переопределения, чтобы разрешить долгосрочное использование ресурсов в тестовых средах.

Как обеспечить тегирование даты истечения срока действия по умолчанию, но освободить некоторые ресурсы от истечения срока жизни? Вы можете создать тест мягкой политики. Мягкая полити-

ка создает исключения и проверяет долгосрочные ресурсы в средах тестирования.

Давайте напишем тест, который применяет тег истечения срока жизни для каждого серверного ресурса, как показано в листинге 12.11. Если сервер не существует в списке исключений, он не проходит тест и не позволяет конвейеру доставки развернуть все изменения в рабочей среде.

Листинг 12.11 Тест для проверки, есть ли у ресурсов срок жизни

```
import pytest

def test_all_nonprod_resources_should_have_expiration_tag(
    servers, server_exemptions):
    noncompliant = []
    for name, values in servers.items():
        if 'expiration' not in values['labels'].keys() and \
            name not in server_exemptions:
            noncompliant.append(name)
    assert len(noncompliant) == 0, \
        'all nonprod resources should have ' + \
        f'expiration tag, {noncompliant}'
```

- ❶ Получаем список серверов в конфигурации и серверов, исключенных из политики.
- ❷ Проверяем, существует ли тег с истекающим сроком жизни в тегах сервера.
- ❸ Если этот сервер не входит в список исключений, помечаем его как не соответствующий политике (не прошедший тест).

Добавление ресурса в список исключений означает, что ваши коллеги по команде будут внимательно проверять, какие ресурсы сохраняются в тестовой среде. Во время коллективной экспертной оценки (глава 7) вы можете определить любой новый постоянный ресурс, добавляя его в список исключений. Единый источник постоянных ресурсов в тестовой среде гарантирует, что вы сможете проводить аудит и обсуждать контроль затрат на ранних этапах процесса разработки.

После реализации тега срока жизни в IaC вам необходимо написать сценарий, который будет выполняться ежедневно. На рис. 12.10 показан рабочий процесс сценария. Он проверяет, совпадает ли дата истечения срока жизни ресурса с текущей датой. Если да, то сценарий автоматически удаляет ресурс.

Зачем устанавливать дату истечения срока жизни с помощью IaC? Рабочий процесс установки даты с помощью модуля тегов предусматривает возможность простого продления срока жизни ресурса! Вместо того чтобы создавать для этого отдельный инструмент автоматизации, вы органично встраиваете обновления в процесс разработки.

Например, если команде по-прежнему нужен ресурс, она всегда может перезапустить конвейер поставки IaC, чтобы сбросить дату

истечения срока действия еще на семь дней. Активные изменения ресурса также сбрасывают срок жизни. Если вы измените свою инфраструктуру, вам, вероятно, все равно понадобится этот ресурс.

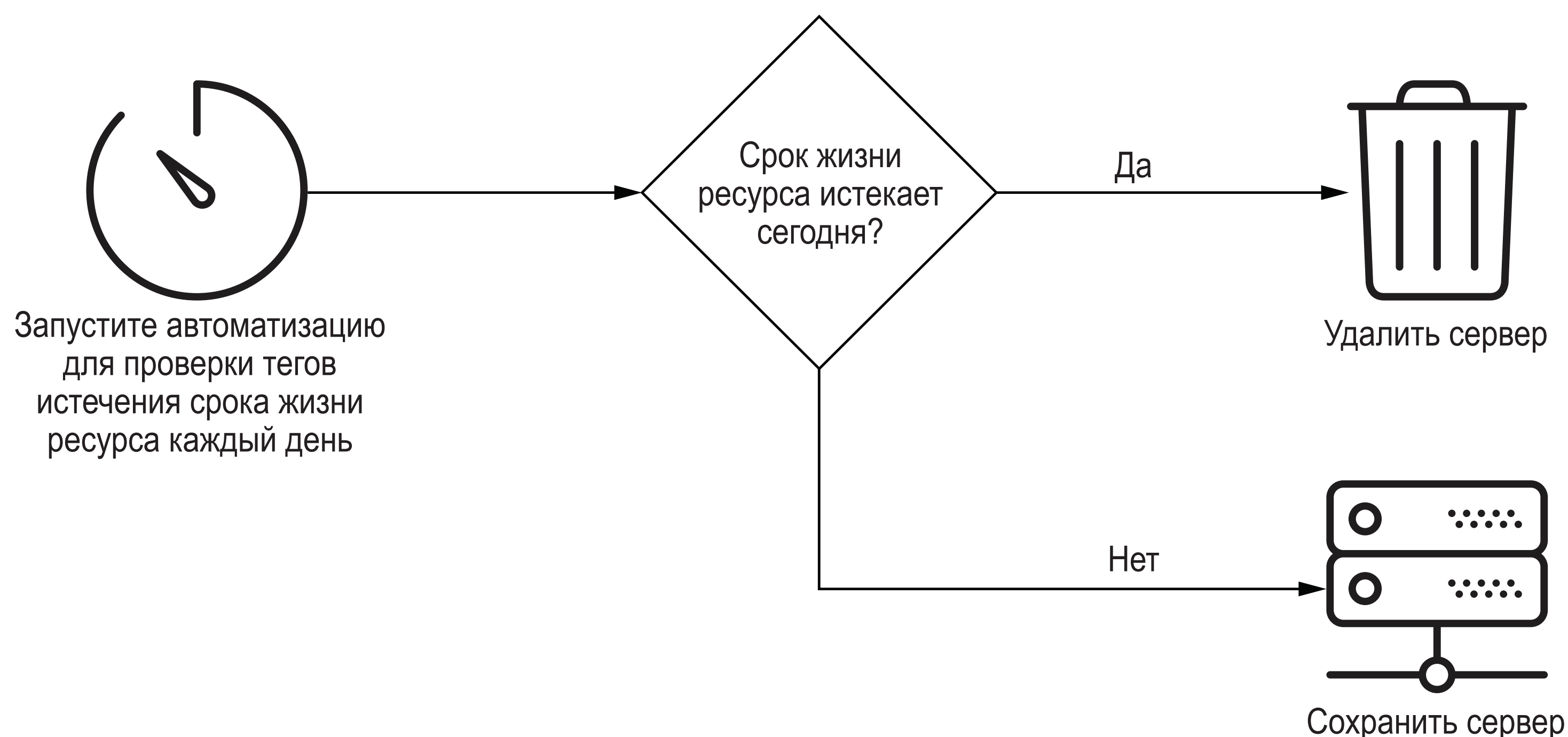


Рис. 12.10 Назначение тега срока жизни позволяет автоматизировать ежедневную проверку необходимости удаления временного ресурса и, таким образом, снизить затраты

Что происходит, когда срок жизни ресурса истекает, а он вам все еще нужен? Вы всегда можете перезапустить свой IaC и создать новый. Использование IaC для добавления и продления срока действия обеспечивает соответствие затрат ожиданиям и функциональную прозрачность для всех команд.

ПРИМЕЧАНИЕ Иногда для тегирования применяют отдельные инструменты автоматизации. Автоматизация добавляет срок жизни к ресурсам инфраструктуры после их создания. Хотя автоматическое тегирование обеспечивает более строгий контроль над соблюдением затрат, оно также приводит к расхождению между фактической и предполагаемой конфигурациями. Более того, автоматическое назначение срока жизни часто сбивает с толку членов команды. Если они не обратят внимания на сообщения системы, то через несколько дней могут обнаружить, что их ресурс внезапно исчез!

Вы всегда можете установить продолжительность жизни ресурса, отличную от заданного количества дней. Если вы хотите обеспечить командам возможность выбора, предложите им диапазон дней с помощью модуля тегов. Я рекомендую также рассчитать абсолютную дату истечения срока жизни и добавить ее в тег вместо временного интервала для упрощения автоматизации очистки.

Интересно, какими стали расходы вашего клиента после реализации всех изменений? Сумма ежемесячного счета уменьшилась с 10 000 до примерно 6500 долл. (снижение расходов на 35 %)! Ваш

клиент доволен эффективным использованием своих облачных ресурсов.

В действительности вы можете и не добиться такого значительного снижения затрат, как в примере. Однако вы всегда можете применить эти практики и методы к своей IaC, чтобы внести незначительные изменения, которые, когда это возможно, снизят затраты. Постоянный контроль источников затрат в IaC заставляет разработчиков помнить о стоимости ресурсов, которые они закладывают в свои решения.

Упражнение 12.2

Представьте, что у вас есть три сервера. Вы изучаете их использование и замечаете следующее:

- вам нужен один сервер для обслуживания минимального трафика;
- для обслуживания максимального трафика вам нужны три сервера;
- сервер обрабатывает трафик 24 часа в сутки, семь дней в неделю.

Что можно сделать, чтобы оптимизировать стоимость серверов на следующий месяц:

- a запланировать остановки ресурсов на выходные;
- b добавить `autoscaling_policy` для масштабирования в зависимости от памяти;
- c установить для всех серверов срок жизни три часа;
- d перенести серверы на машины с меньшим количеством процессоров и памяти;
- e перенести приложение в контейнеры и плотнее сжать их на серверах.

Ответы на упражнения приведены в приложении В.

12.3 *Оптимизация затрат*

Для сокращения облачных излишков и управления факторами затрат можно применить и другие методы IaC, кроме упомянутых ранее. Такие методы, как создание сред по требованию, обновление маршрутов между регионами или тестирование в производстве, могут опираться на технологии IaC для дальнейшей оптимизации затрат, как показано на рис. 12.11.

В частности, принципы воспроизводимости, компоуемости и развиваемости способствуют поиску творческих методов дальнейшей оптимизации затрат. Эти методы включают в себя воспроизведение сред по требованию для отказа от постоянных тестовых сред, объединение инфраструктуры провайдеров облачных услуг и развитие производственной инфраструктуры, охватывающей различные регионы и провайдеров облачных услуг.



Рис. 12.11 Оптимизация затрат требует применения методов IaC для масштабирования и развертывания ресурсов инфраструктуры

В примере этой главы вы сократили затраты вашего клиента на облачные вычисления для конференции на 35 %. Год спустя финансовый отдел просит вас помочь оптимизировать стоимость их платформы. Они расширили свой бизнес и хотят оптимизировать затраты на сотни клиентов и управляемые услуги.

12.3.1 Создание сред по требованию

В более широкой перспективе вам необходимо изучить, какие среды существуют у компании в сегментах тестирования и производства. Вы можете начать с сокращения облачных излишков во всех средах. Однако по мере роста вашей компании она будет добавлять все больше сред для поддержки и тестирования большего количества продуктов.

Представьте, что вы изучаете инфраструктуру вашего клиента. Клиент имеет много тестовых сред. Вы видите, что три или четыре из них работают непрерывно и предназначены для специализированного тестирования. Например, команда обеспечения качества (quality assurance, QA) использует одну из сред для тестирования производительности два раза в год. В остальное время эти среды простаивают.

Вы решили *удалить постоянно работающие среды*. Если команде QA нужна среда для тестирования производительности, она может получить ее по требованию. Команда копирует производственную среду с модулями фабрик и строителей образов, которые допускают ввод данных конфигурации. Модули обеспечивают гибкую настройку переменных и параметров для различных сред.

На рис. 12.12 показана остальная часть рабочего процесса создания среды по требованию. Команда QA копирует IaC в новый репозиторий, предназначенный для тестовой среды в структуре репозитория организации. Команда обновляет параметры и переменные, выполняет тесты и удаляет среду.

Зачем соблюдать принцип воспроизводимости при создании новых сред по требованию с последующим удалением? Вновь созданная среда гарантирует соответствие последней конфигурации производственной среды. Если вы используете среду только один

раз в год, вам не нужно, чтобы она работала постоянно в течение 11 месяцев.

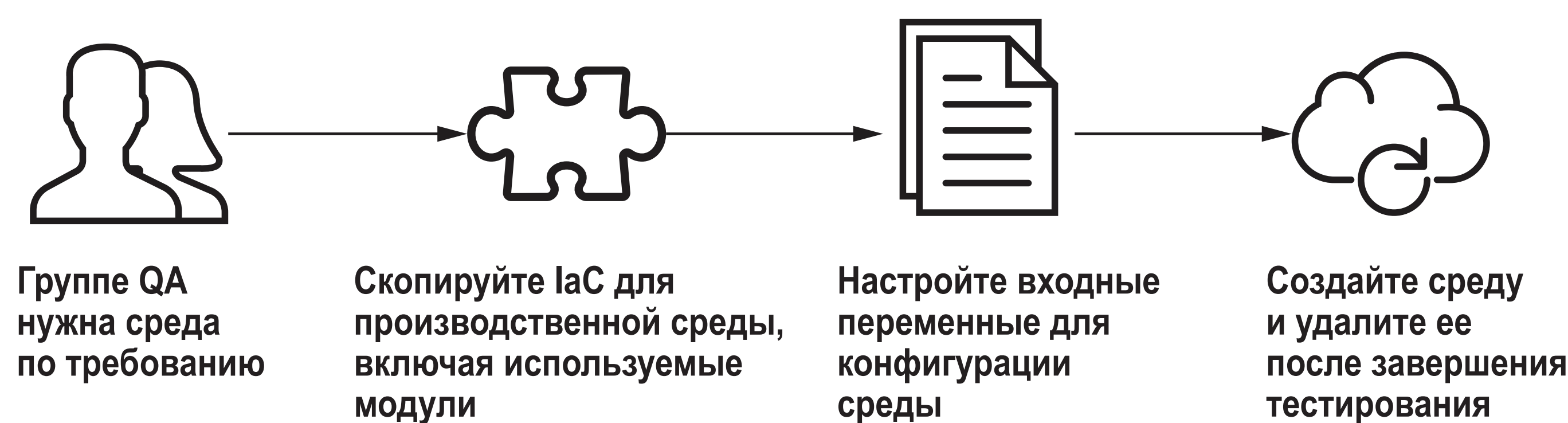


Рис. 12.12 Вы можете скопировать конфигурацию производственной среды, чтобы создавать и настраивать тестовые среды по мере необходимости

Хотя создание новой среды требует времени, вы, вероятно, потратите столько же времени, пытаясь исправить расхождение между тестовой и производственной средами. Выявление ненужных долгоживущих сред и перенос их на модель по требованию помогает снизить затраты, особенно если вы можете легко их воссоздать.

12.3.2 Использование нескольких облаков

Имея несколько вариантов поставщиков облачных услуг, вы также можете рассмотреть возможность развертывания в других облаках и оптимизации затрат в зависимости от ресурсов, рабочей нагрузки и времени суток. IaC может помочь стандартизировать и организовать конфигурацию для нескольких облаков. Развертывание в нескольких облаках может удовлетворить специализированные рабочие нагрузки или команды, которым требуются определенные ресурсы инфраструктуры.

Например, представьте, что ваш клиент использует Google Cloud Dataflow для потоковой обработки данных. Однако стоимость варьируется в зависимости от типа конвейера. Вы убеждаете некоторые команды преобразовать несколько конвейеров пакетной обработки в Amazon EMR, чтобы снизить общие затраты.

Эквивалент Azure

Эквивалентом Amazon EMR или Google Cloud Dataflow является Azure HDInsight.

На рис. 12.13 показано, что команда службы отчетности переключает свой IaC на использование модуля Amazon EMR. Чтобы свести к минимуму сбои в выполнении заданий, члены группы используют blue-green схему развертывания из главы 9 и постепенно увеличивают количество заданий, выполняемых ими в Amazon EMR.

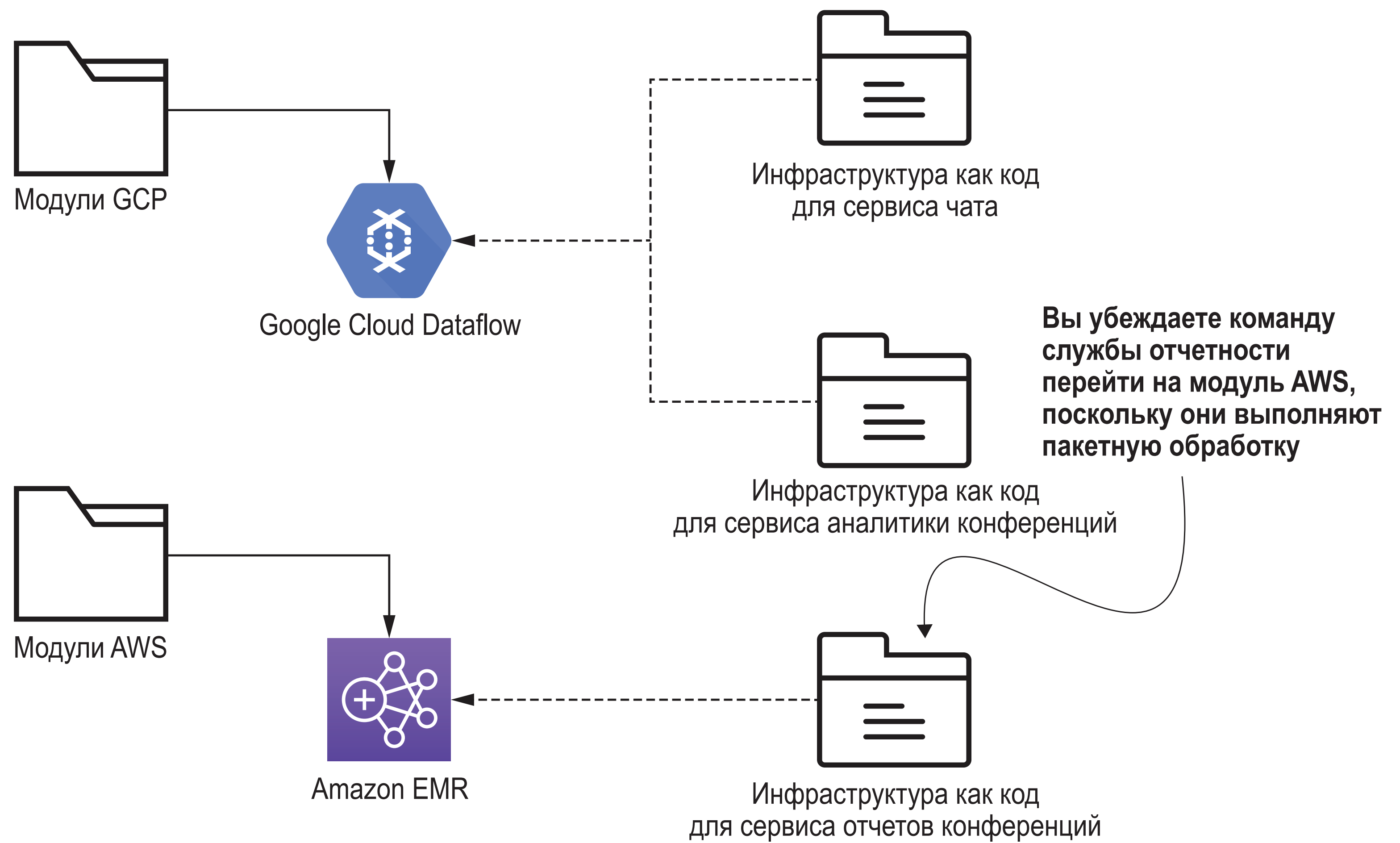


Рис. 12.13 Служба отчетности переключает задание пакетной обработки на использование Amazon EMR вместо Google Cloud Dataflow, используя другой модуль

Принцип компоновки становится важной частью мультиоблачной конфигурации. IaC упрощает управление и различение ресурсов инфраструктуры различных провайдеров облачных услуг. Использование модулей для выражения зависимостей между облаками облегчает дальнейшее развитие ресурсов.

В главе 5 мы разделили конфигурацию IaC на разные каталоги в зависимости от инструментов и провайдеров. Многие инструменты IaC не предусматривают единую модель данных для ресурсов провайдера облачных услуг. Создайте отдельный модуль для каждого облака, которое вы планируете поддерживать. Разделение модулей по провайдерам снижает сложность и упрощает изолированное тестирование модулей. Наличие отдельных модулей для каждого поставщика облачных услуг также позволяет легко различать ресурсы и провайдеров инфраструктуры.

12.3.3 Оценка трафика между регионами и облаками

После перехода на несколько облаков может оказаться, что общий счет за облачные вычисления не уменьшился. Вам необходимо тщательно оценить целесообразность использования нескольких облаков, поскольку провайдеры взимают плату за передачу данных между регионами и за пределами своей облачной сети. Совокупные затраты на передачу данных могут вас неприятно удивить!

Вы анализируете счет вашего клиента за облачные вычисления и замечаете, что большая часть затрат приходится на передачу данных между регионами и за пределы сети. После некоторого рас-

следования выясняется, что многие службы и среды тестирования обмениваются данными между регионами, зонами доступности и через общедоступный интернет.

Например, интеграционные тесты службы чата в `us-central1-a` используют общедоступный IP-адрес службы профилей пользователей в `us-central1-b`! Очевидно, что интеграционные тесты служб не следует выполнять в разных регионах, зонах доступности или за пределами частной сети.

Интеграционное тестирование проверяет функциональность службы только относительно других служб, а не самой системы. На рис. 12.14 показано применение методов рефакторинга из главы 10 для консолидации ресурсов инфраструктуры в среде интеграционного тестирования в одну зону доступности.

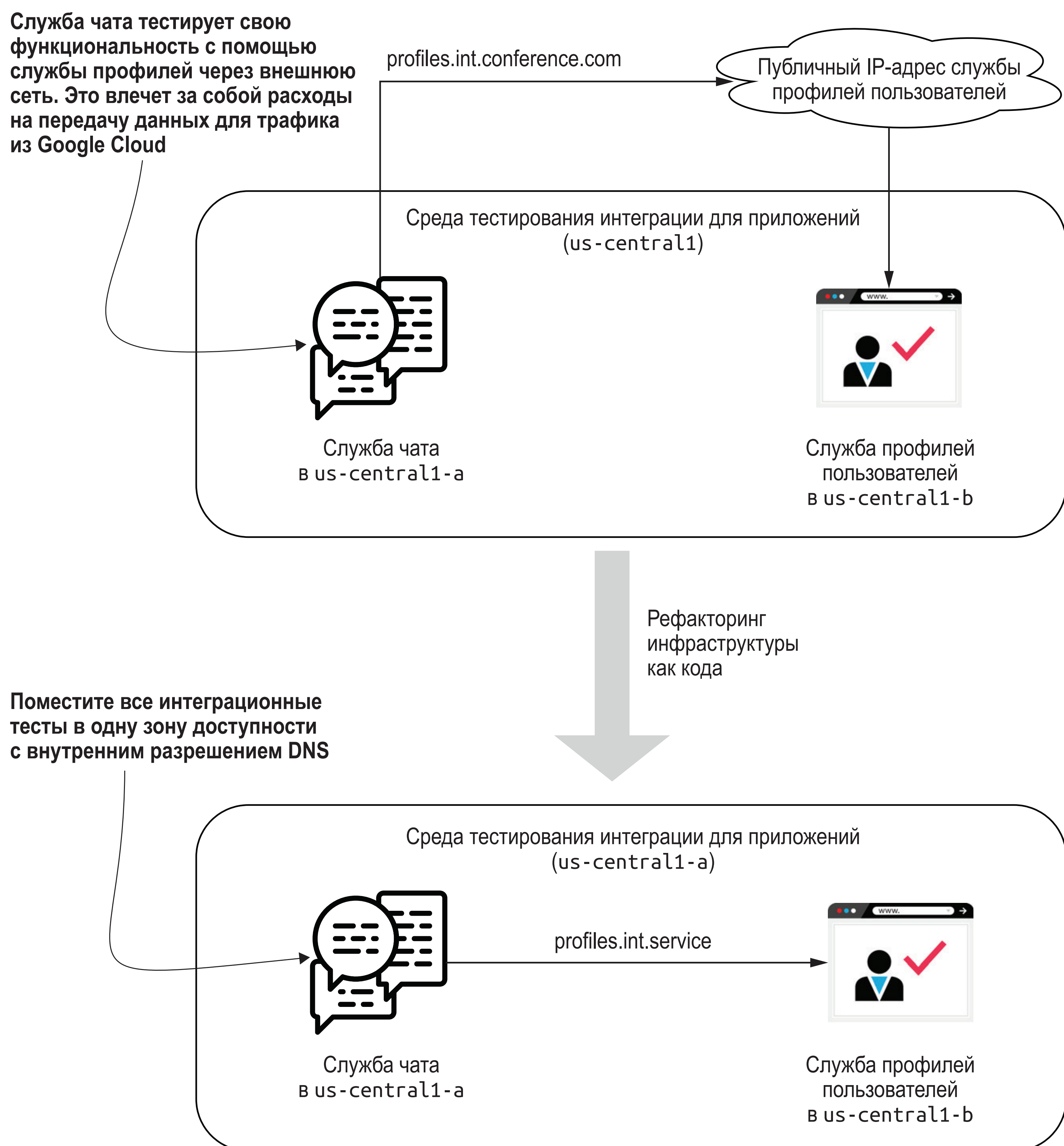


Рис. 12.14 Рефакторинг вашей IaC для поддержки среды интеграционного тестирования в единой зоне доступности и в диапазоне частных IP-адресов

Что произойдет, если зона доступности выйдет из строя? Вы всегда можете целиком переключить IaC в другую зону или регион. Приложения продолжают обмениваться данными через частную сеть, и провайдер не будет взимать плату за передачу данных из сети Google Cloud или между регионами и зонами.

Частные сети не только безопаснее, но и выгоднее с точки зрения расходов на внешний трафик. Если вы используете несколько облаков, выясните, какие ресурсы должны обмениваться данными между облаками. Иногда выгоднее перенести весь набор услуг в другое облако, чем платить за передачу данных между облаками. Воспользуйтесь методами изменения и рефакторинга, описанными в главах 9 и 10, чтобы консолидировать службы и коммуникации.

12.3.4 Тестирование в производстве

Даже после перехода на несколько облаков и оптимизации передачи данных вы можете обнаружить, что ваша тестовая среда не полностью имитирует производственную среду и ее эксплуатация обходится слишком дорого. В какой-то момент окажется, что вам недостаточно тестирования в изолированной среде. Вместо полного моделирования производственной среды можно оптимизировать затраты, выполняя часть тестов непосредственно в производственной среде.

В случае с конференц-платформой вы помогаете команде видеосервиса внедрять изменения для тестирования в производстве. На рис. 12.15 показано, что члены команды создают новый набор ресурсов инфраструктуры, при этом ожидаемые изменения скрыты за флагом функции. Затем они переключают флаг, чтобы направить все приложения и пользовательский трафик на обновленные

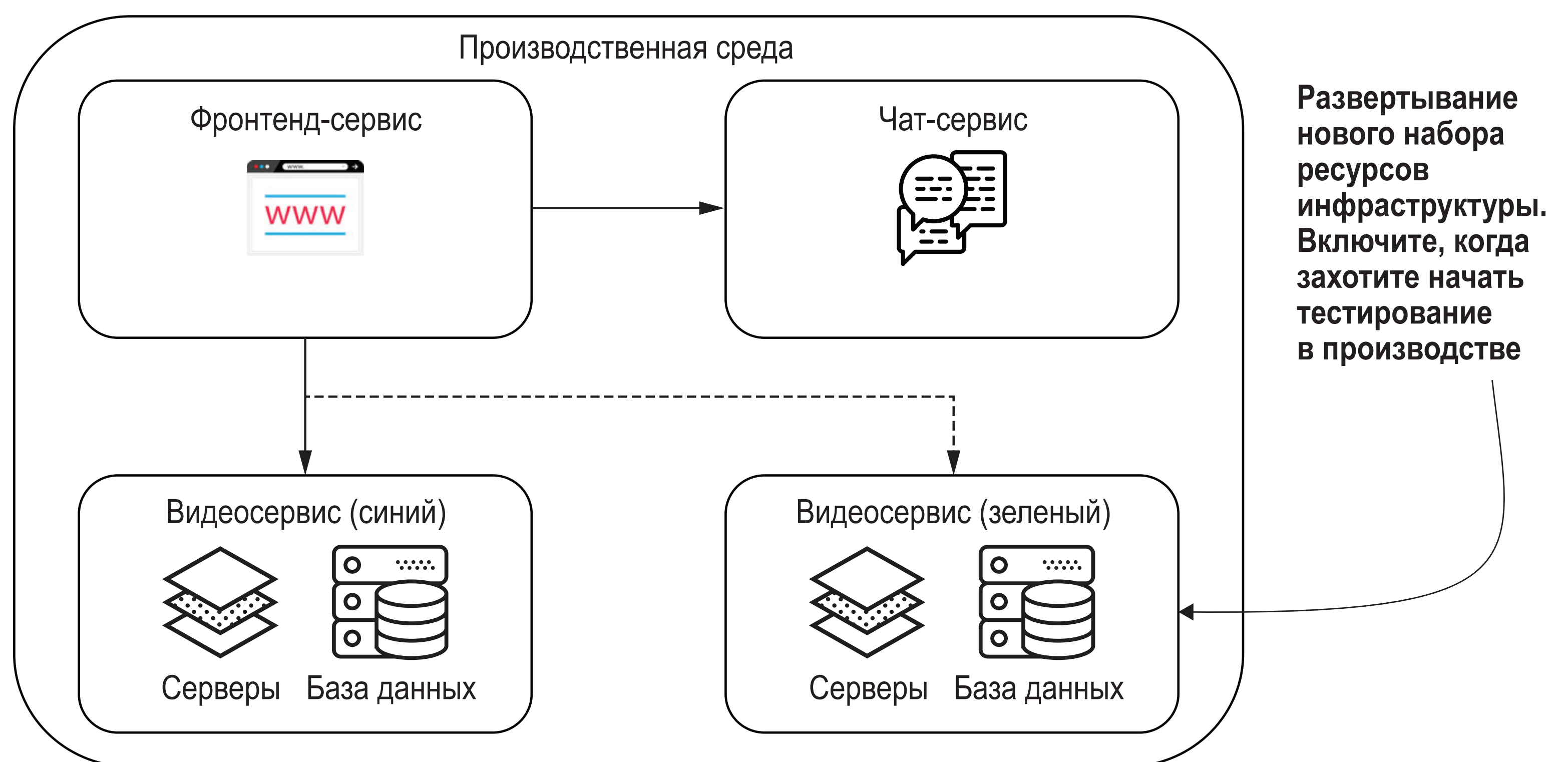


Рис. 12.15 При производственном тестировании IaC применяются blue-green развертывания и пометка функций для набора ресурсов

ресурсы и проверить их функциональность в рабочей среде. Два набора ресурсов работают одновременно в течение нескольких недель. Через несколько недель команда удаляет старые ресурсы инфраструктуры.

Команда протестировала свой сервис в производственной среде, используя blue-green развертывание без привлечения тестовой среды. Тестирование в производственной среде основано на методике безопасного запуска тестов на производственных данных и системах.

ОПРЕДЕЛЕНИЕ *Тестирование в производстве* (testing in production) – это набор практик, которые позволяют запускать тесты на производственных данных и системах.

При разработке программного обеспечения применяются такие методы, как флаги функций, позволяющие скрывать определенные функциональные возможности в рабочей среде для тестирования. Вы уже знаете, что канареечное развертывание традиционно применяется для тестирования функциональности ПО на небольшой группе пользователей, прежде чем делать ее общедоступной.

В случае IaC тестирование в производстве не полностью аналогично стандартной практике разработки ПО. Вас не интересует, работает ли код у небольшой группы пользователей. Вы хотите знать, не сломает ли ваше обновление инфраструктуру системы и не повлияет ли этот сбой на широкий круг пользователей! В вашем распоряжении есть несколько методов, таких как флаги функций и канареечное развертывание. Мы обсуждали их в главах 9 и 10.

Вы можете протестировать IaC непосредственно в производственной среде без blue-green развертывания или флагов функций. Но у вас должен быть наготове детально проработанный и утвержденный руководством план восстановления в случае сбоя. Я работала в одной организации, которая строго придерживалась методики локального тестирования, прежде чем вносить изменения в производство. Если наше изменение не удалось, мы попытаемся обновить систему до предыдущего состояния. Если и это не помогло, остается лишь создать совершенно новую среду и направить в нее весь трафик приложений и пользователей.

Вы можете последовательно применить все методы контроля и оптимизации затрат и сокращения облачных излишков, но так и не достичь полной оптимизации расходов! Со временем требования к технологиям и продуктам вашей организации меняются.

Если вы регулярно применяете инструменты мониторинга системы, то можете обнаружить периоды повышенного или пониженного спроса. Например, платформа вашего клиента пользуется наибольшим спросом в мае, июне, октябре и ноябре, в пик сезона конференций.

ПРИМЕЧАНИЕ Возможно, вам придется перепроектировать свои системы, чтобы воспользоваться эластичностью

общедоступного облака – возможностью увеличения или уменьшения масштаба и снижения затрат в разные периоды времени. Некоторые архитектуры плохо поддерживают динамическое масштабирование ресурсов. Для масштабирования с целью повышения экономической эффективности системы часто требуется рефакторинг или полная замена платформы приложения.

Анализ использования ресурсов вашей системы и предъявляемых к ней требований в разные периоды времени поможет вам в дальнейшей оптимизации затрат, помимо методов, упомянутых в этой главе. Вы можете еще больше сократить расходы, заключив контракт с провайдером инфраструктуры. Выбор новой модели ценообразования позволяет сэкономить на определенном количестве зарезервированных экземпляров или получить скидки в зависимости от объема.

Чем дольше вы используете систему, тем больше показателей вы собираете. Опираясь на эти данные, вы будете разрабатывать меры по контролю факторов затрат, сокращению облачных излишков и общей оптимизации затрат в IaC. Эта информация пригодится также во время переговоров с провайдером облачных услуг и поможет избежать непредвиденных ситуаций при оплате счетов за облачные вычисления!

Краткое содержание главы

- Прежде чем вносить в IaC изменения для оптимизации затрат, определите факторы затрат (ресурсы или действия), наиболее сильно влияющие на общую стоимость.
- Управляйте факторами затрат в инфраструктуре, например вычислительными ресурсами, путем добавления тестов политики для проверки типа, размера и срока резервирования ресурсов.
- Оценка стоимости представляет собой анализ атрибутов ресурсов IaC и расчет затрат на основе ответа API провайдера облачных услуг.
- Тестирование ресурсной политики направлено на проверку результатов оценки затрат и определение того, не превышен ли бюджет.
- Облачные излишки – это неиспользуемые или недостаточно используемые ресурсы инфраструктуры.
- Устранение облачных излишков способствует снижению затрат на облачные вычисления.
- Для сокращения облачных излишков удалите или остановите непомеченные или неиспользуемые ресурсы, запускайте и останавливайте ресурсы по расписанию, правильно определяйте размеры

ресурсов инфраструктуры для более эффективного использования, включите автоматическое масштабирование и помечайте дату истечения срока жизни ресурса.

- Автомасштабирование увеличивает и уменьшает количество или объем ресурсов в определенной группе на основе данных о потребности в таких ресурсах, как ЦП или память.
- К методам оптимизации затрат на облачные вычисления относятся создание сред по требованию, использование нескольких облаков, оценка объема передачи данных и тестирование в производстве.
- При тестировании изменений инфраструктуры непосредственно в производственной среде применяются такие методы, как blue-green развертывание и флаги функций.

15

Выбор и обновление инструментов

Основные темы главы:

- оценка использования модулей и инструментов инфраструктуры с открытым исходным кодом;
- применение методов обновления или миграции инструментов IaC;
- реализация паттернов модулей для управляемой событиями IaC.

Вы узнали, как разрабатывать инфраструктуру, представленную в виде кода, обновлять ее совместно с остальными членами команды, используя конвейеры доставки и тестирование, а также управлять ее безопасностью и затратами внутри своей организации. По мере развития вашей инфраструктурной системы вы будете осваивать новые приемы и методы и адаптировать их к новым рабочим процессам и вариантам использования. Инструменты тоже меняются, поэтому нужно позаботиться о том, чтобы изменения инструментов не нарушали масштабирование, совместную работу и эксплуатацию вашей инфраструктуры.

Существуют разные способы обновления инструментов. Вы можете перейти на новую версию, новый инструмент или рассмотреть другие, более динамичные варианты использования IaC. В этой гла-

ве обсуждаются общие паттерны и методы обработки обновлений инструментов IaC.

Эти паттерны применимы к любым инструментам, предназначенным для начального конфигурирования, управления конфигурацией и создания образов. Нужно признать, что они изначально применялись к разработке обычного ПО, а я довольно смело адаптировала их к разработке инфраструктуры. Используйте эти паттерны и методы, чтобы минимизировать потенциальный ущерб от сбоя обновлений, масштабировать новый инструмент по мере роста потребностей команд и продолжать развивать свою систему в соответствии с потребностями бизнеса.

ПРИМЕЧАНИЕ В этой главе не приводятся листинги кода. Для добавления примеров пришлось бы рассматривать новые инструменты. Я описываю закономерности и подходы на более высоком уровне. Вы можете применить эти методы к любому инструменту, поддерживающему DSL или обычные языки программирования.

Вы прочитали все главы этой книги и применяли IaC в различных отраслях и компаниях. Вы заработали репутацию специалиста по настройке и масштабированию конфигураций IaC. Однажды компания, занимающаяся социальными сетями, предлагает вам войти в команду своей платформы.

Компания уже несколько лет использует технологии IaC. Сотрудникам нужна ваша помощь в обслуживании и обновлении инструментов, которые они применяют при разработке и поддержке IaC. Вы принимаете предложение и получаете портфель проектов, в которые нужно погрузиться с первого дня работы.

13.1 *Использование инструментов и модулей с открытым исходным кодом*

Благодаря доступности систем контроля версий и открытых репозиторий часто бывает проще найти готовый инструмент или модуль инфраструктуры вместо написания собственного. Вы можете зайти на GitHub или любой аналогичный сервис, чтобы скачать средства автоматизации и инструменты, соответствующие вашим потребностям. Однако вам необходимо убедиться, что вы проявили должную осмотрительность, прежде чем внедрять новые инструменты в своей организации.

Например, предположим, что к вам обратились члены команды, которые поддерживают функциональность ленты социальных сетей. Они нашли в интернете инфраструктурный модуль для создания базы данных и хотят его использовать. Они надеются ускорить

процесс разработки и не ждать, пока другая команда проверит конфигурацию их базы данных. В конце концов, зачем изобретать велосипед?

Вы предлагаете им свою помощь в проверке модуля на предмет безопасности и соответствия современным технологиям. Прежде чем представить модуль другим командам компании, занимающейся социальными сетями, вы выполняете проверку по схеме, показанной на рис. 13.1, чтобы оценить функциональность, безопасность и жизненный цикл модуля базы данных, прежде чем рекомендовать использовать его в повседневной работе.

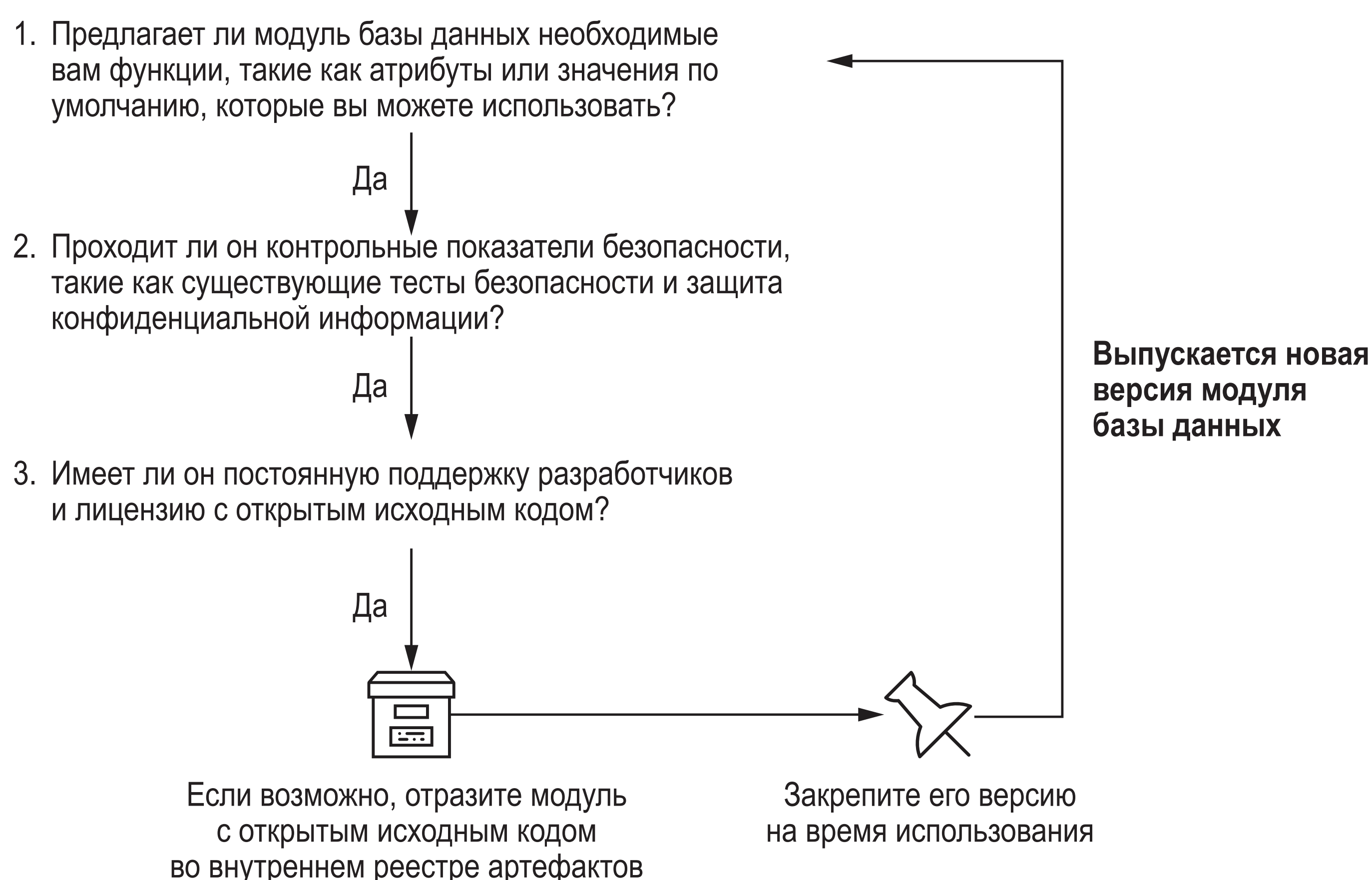


Рис. 13.1 Проверка функциональности, безопасности и жизненного цикла инструмента или модуля перед его использованием

Каждый раз, когда разработчики открытого исходного кода выпускают новый модуль базы данных, вы заново проверяете этот модуль. Вы можете применить этот рабочий процесс принятия решений для безопасного и надежного внедрения внешних модулей и инструментов IaC. Вы должны убедиться, что применение этого модуля не приведет к появлению уязвимости инфраструктуры, которая позволит злоумышленнику проникнуть в вашу систему.

13.1.1 Функциональность

Существуют модули и инструменты, готовые к началу работы. Они позволяют очень гибко настраивать нужные вам атрибуты. Но в главах 2 и 3 было сказано, что модули должны иметь некоторые важные значения по умолчанию. Без них модуль или инструмент со слишком большой гибкостью может привести к созданию причудливых конфигураций, которые в конечном итоге сломают вашу систему.

Вы рекомендуете команде по работе с лентами новостей проверить значения по умолчанию в модуле базы данных. Команда оценивает модуль, как показано на рис. 13.2. Модуль использует очень строгие значения по умолчанию, определяет версию базы данных и тщательно проверяет совместимость.

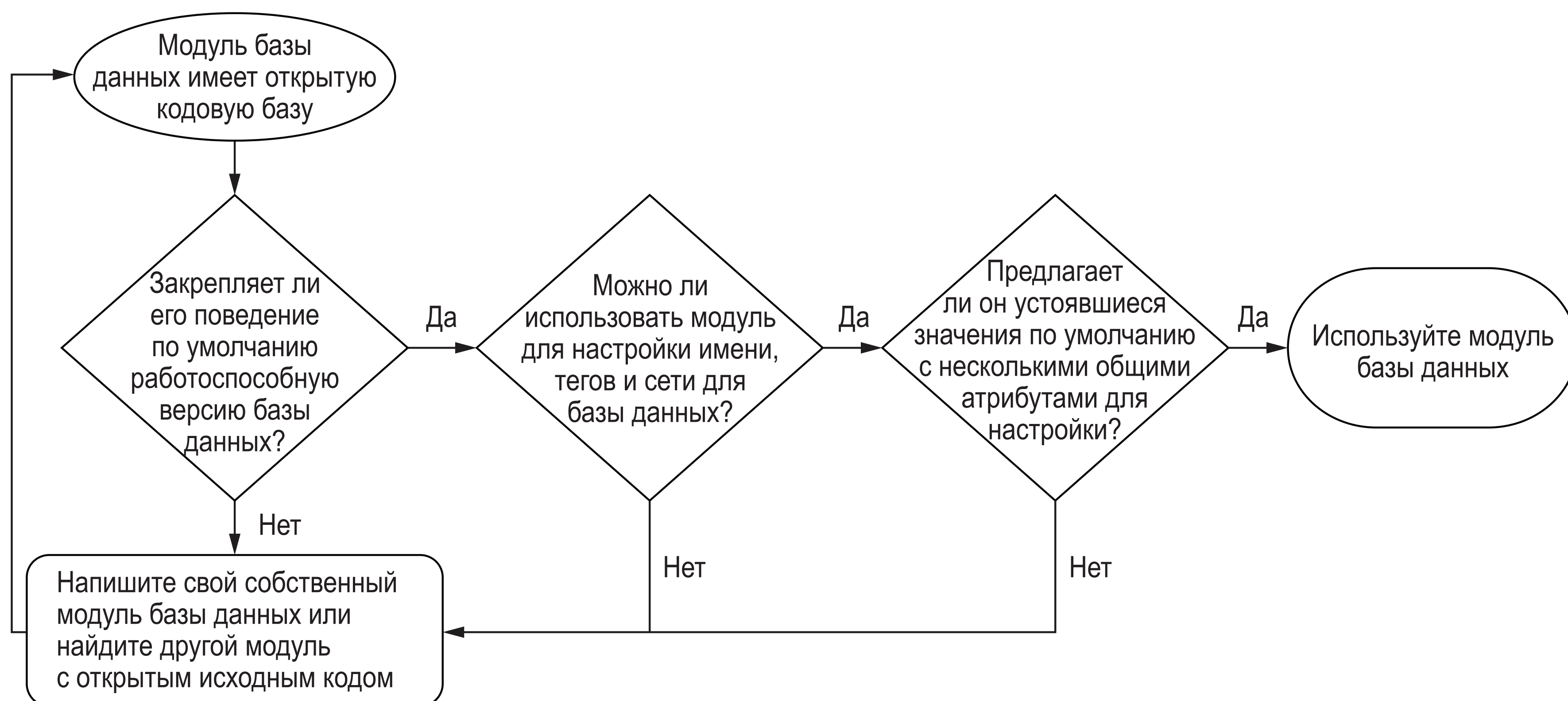


Рис. 13.2 Схема принятия решения для оценки использования модуля или инструмента на основе функциональности

В документации отмечается, что модуль фиксирует версию базы данных для тщательного тестирования конфигурации и ее совместимости с конкретными версиями. Члены команды новостных лент подтверждают, что они используют нужную версию базы данных, и соглашаются с настройками модуля по умолчанию.

Затем они проверяют входные переменные для модуля. Модуль базы данных позволяет им настраивать необходимые атрибуты, например имя базы данных. Он также позволяет им задавать теги и сеть. Члены команды новостных лент подтверждают, что их устраивает имеющийся набор переменных.

Поскольку модуль не получает каждый атрибут в качестве входной переменной и использует важные значения по умолчанию, которые подходят команде, вы одобряете модуль преимущественно по его функциональности. Начните с просмотра документации модуля и истории коммитов. Если модуль проверяет совместимость версий и набор определенных значений по умолчанию, которые соответствуют вашим функциям, вы можете перейти к оценке безопасности.

Если вы обнаружите, что модуль не содержит нужных значений по умолчанию или входных переменных, которые необходимо изменить, вы можете найти другой модуль, написать свой собственный или начать использовать модуль с учетом его ограничений. Точно так же вы можете проанализировать текущий инструмент и обнару-

жить, что его функциональности недостаточно. Один инструмент не может сделать все! При выборе инструмента старайтесь соблюдать баланс между его функциональной гибкостью и предсказуемостью при внесении изменений в инфраструктуру.

13.1.2 Безопасность

Вы начинаете выбор модуля или инструмента с анализа функциональности, но сразу после этого следует перейти к оценке безопасности. На самом деле именно безопасность служит решающим критерием того, следует вам использовать инструмент или модуль с открытым исходным кодом. Без тщательной оценки модуля или инструмента с открытым исходным кодом на предмет безопасности конфигурации или кода вы можете непреднамеренно оставить злоумышленнику лазейку для компрометации системы.

Прежде чем члены команды новостных лент смогут использовать свой модуль базы данных, вы должны проверить модуль на наличие проблем с безопасностью. Вы смотрите, предоставляет ли модуль базы данных или выводит конфиденциальную информацию, отправляет ли он информацию на сторонние http-адреса и проходит ли существующие тесты безопасности и соответствия, как показано на рис. 13.3.

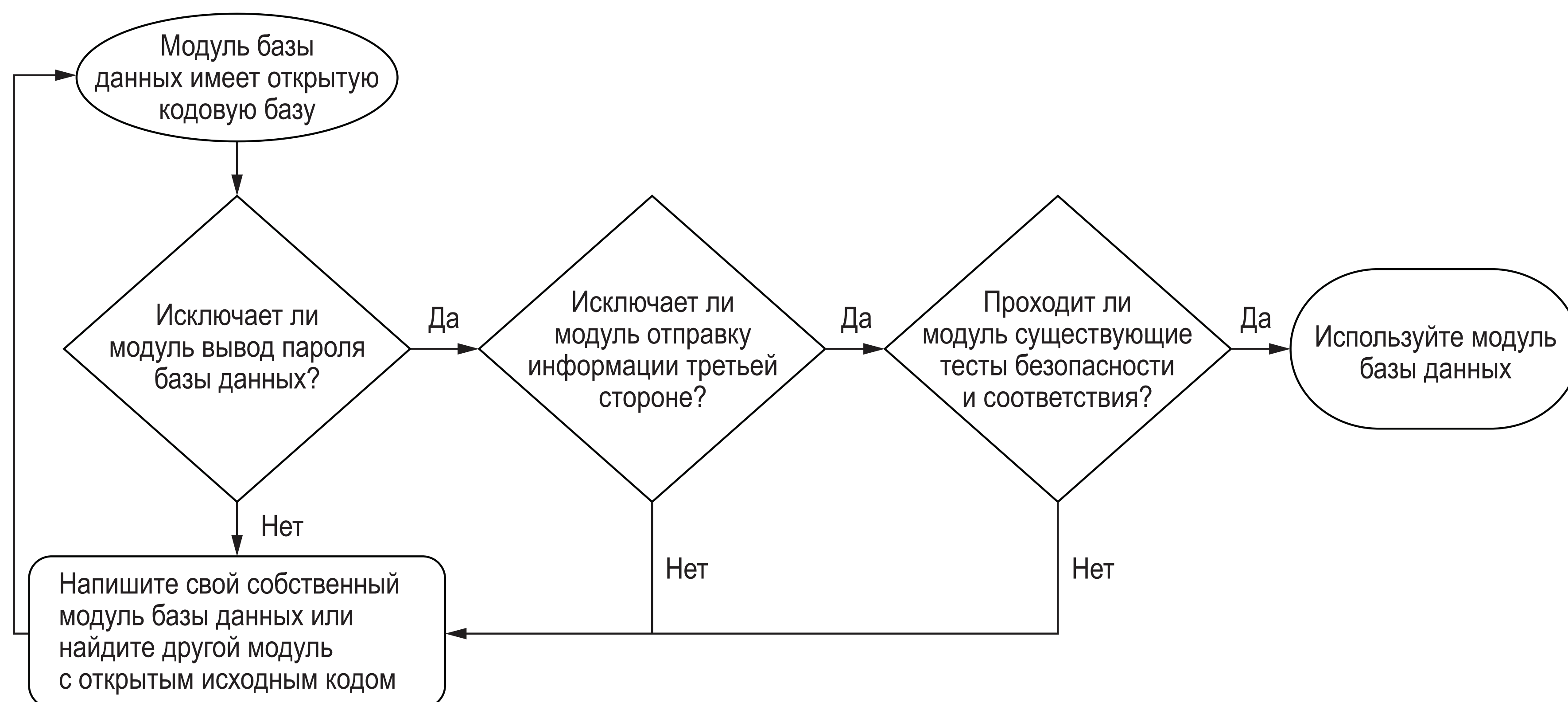


Рис. 13.3 Схема принятия решения об использовании модуля или инструмента с точки зрения безопасности

В этом примере модуль базы данных не выводит пароль или какую-либо конфиденциальную конфигурацию и не отправляет информацию сторонней конечной точке. Модуль также проходит тесты безопасности и соответствия для баз данных, которые вы написали в главе 8. Почему вам следует выполнять все три проверки перед внедрением модуля?

Модуль может случайно раскрыть или вывести конфиденциальную информацию. Например, конфигурация может случайно вывести пароль во время пробного прогона. Если это так, убедитесь, что у вас есть способ смягчить последствия или замаскировать пароль и впоследствии изменить его.

Аналогично модуль не должен записывать или отправлять информацию неавторизованной третьей стороне. Злоумышленник может добавить малозаметную деталь конфигурации, которая отправляет информацию о вашей сети на внешний HTTP-адрес. Просмотрите каждый ресурс и убедитесь, что он ничего не отправляет третьим лицам.

Безопасность и открытый исходный код

Атаки в цепочке поставок происходят, когда злоумышленник включает вредоносный код в поставляемое ПО, которое отправляется клиентам и ставит под угрозу их данные и системы. Преимущество открытого исходного кода заключается в возможности тщательно изучить его, прежде чем приступить к использованию.

Надо признать, что в этом разделе я сильно упростила оценку рисков и меры защиты от атак в цепочке поставок. Для получения дополнительной информации ознакомьтесь с официальным документом NIST (<http://nng.bz/449B>).

Наконец, выполните внутренние тесты безопасности и совместимости, которые вы написали для ресурсов инфраструктуры. Вам нужны *безопасные* и *совместимые* ресурсы. В противном случае вам придется вернуться к началу и найти новый модуль в соответствии с вашими требованиями.

Разверните модуль в изолированной тестовой среде своей IaC. Затем запустите тесты безопасности и совместимости модуля. Изоляция модуля в тестовой среде гарантирует, что вы не введете несовместимые конфигурации в производственную среду.

Имейте в виду, что некоторые тесты могут быть не пройдены из-за несовместимости самих тестов с новым модулем. В таком случае модифицируйте тесты и повторите тестирование. После прохождения всех тестов можете смело утвердить модуль как безопасный для использования.

Аналогичный рабочий процесс выполняется при анализе безопасности инструментов. Но тестирование безопасности и совместимости инструментов IaC может включать статический анализ кода и дополнительные проверки со стороны вашей организации. Для инструментов, которые выводят конфигурацию или другую информацию во время пробного прогона, вам потребуется применить шаги по исправлению, описанные в главе 8.

13.1.3 Жизненный цикл

Вы проверили функциональность и безопасность модуля, но члены команды комплаенса задают очень хороший вопрос. Они спрашивают, кто обеспечивает сопровождение модуля с открытым исходным кодом. Вашей организации придется заявить частное (или, возможно, публичное) право собственности на модуль или инструмент, если разработчики больше не намерены его обновлять или исправлять.

Чтобы понять жизненный цикл модуля и организацию его поддержки, вам придется изучить документацию, как показано на рис. 13.4. Если модуль базы данных спонсируется компанией и имеет соответствующую лицензию, вы, скорее всего, сможете его использовать.

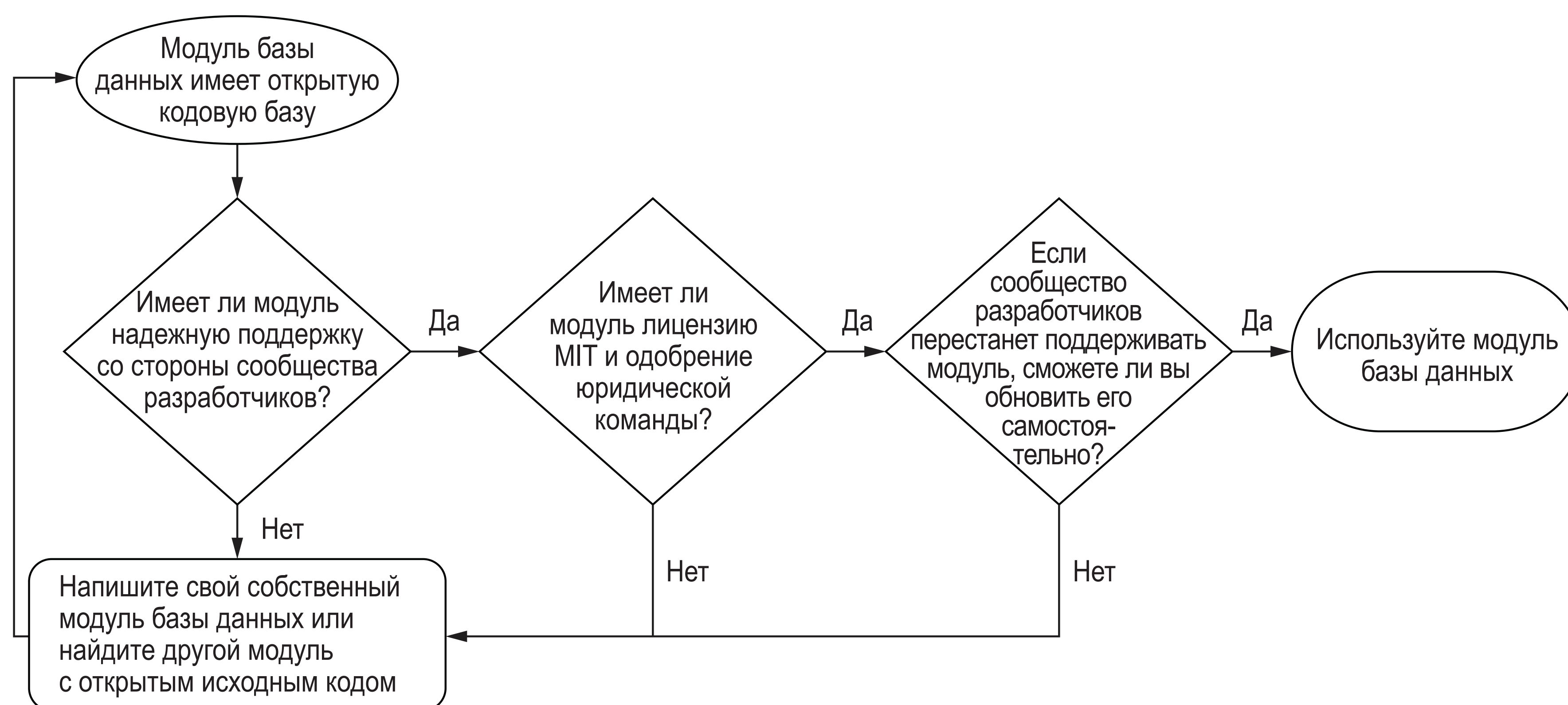


Рис. 13.4 Схема принятия решения для оценки использования модуля или инструмента на основе жизненного цикла инструмента или модуля

Вы проверяете поддержку и сопровождение модуля базы данных. Сопровождением занимаются представители известной технологической компании, а в сообществе разработчиков много участников. Это означает, что проект пользуется вниманием и находится в активном состоянии. Разработчики обновляют и выпускают новую версию модуля каждые несколько месяцев. Вы должны пропустить каждое обновление через набор тестов, чтобы убедиться, что изменения не нарушают работу модуля.

Затем вы получаете информацию о лицензии открытого исходного кода модуля базы данных. Вы не знаете, насколько хорошо эта лицензия подходит для вашей компании, поэтому обращаетесь к своему юридическому отделу. Юристы проверяют лицензию, прежде чем команда новостной ленты сможет ее использовать.

Допустим, нужный вам модуль сопровождается лицензией MIT – разрешительной лицензией ПО с открытым исходным кодом. Ли-

цензия MIT означает, что если вы создаете форк проекта или изменяете модуль базы данных, вы должны включить копию лицензии и оригинальное уведомление об авторских правах. Если разработчики объявляют модуль или инструмент устаревшим, разрешительный тип лицензии позволяет вам самостоятельно изменять и обновлять модуль.

ОПРЕДЕЛЕНИЕ Разрешительная лицензия с открытым исходным кодом позволяет вам ответвлять или изменять код, при условии что вы включаете копию лицензии и оригинальное уведомление об авторских правах.

Ваша команда юристов одобряет лицензию, поскольку модуль представляет минимальный риск для общей конфигурации инфраструктуры. Компания может редактировать модуль при необходимости, но не обязана публиковать его. Возможно, вы даже сможете внести свой вклад в модуль с открытым исходным кодом, пока ожидаете одобрение юристов.

Модуль или инструмент также может включать лицензию с открытым исходным кодом в категории *copyleft* (антикопирайт, отсутствие авторского права). Категория лицензий *copyleft* содержит пункт, согласно которому вы обязаны опубликовать кодовую базу с внесенными вами изменениями.

ОПРЕДЕЛЕНИЕ Лицензия с открытым исходным кодом типа *copyleft* позволяет вам ответвлять или модифицировать код, при условии что вы опубликуете кодовую базу с вашими изменениями.

Категория лицензий *copyleft* часто предусматривает дополнительные ограничения на изменение и распространение инструмента или модуля. Команда юристов должна оценить, может ли компания использовать IaC с открытым исходным кодом с более строгими лицензиями.

ПРИМЕЧАНИЕ Для получения дополнительной информации о лицензиях с открытым исходным кодом ознакомьтесь с лицензиями и стандартами, изложенными Инициативной группой открытого исходного кода (<https://opensource.org/licenses>).

Члены команды новостных лент (они в восторге) могут использовать модуль базы данных. Вы рекомендуете команде зафиксировать версию модуля, разместив ее во внутреннем репозитории артефактов. Внутреннее «замораживание» версии модуля гарантирует, что команды смогут использовать только утвержденный модуль и только во внутреннем репозитории. Если общедоступный

репозиторий модуля выходит из строя, у команды всегда есть копия во внутреннем репозитории. Каждый раз, когда разработчики выпускают новую версию модуля, вы должны проверить изменения, утвердить последнюю версию и закрепить ее во внутреннем репозитории.

Подумайте о возможности вносить свои изменения в открытый исходный код сообщества, если ваша организация это позволяет. Формирование модуля или инструмента с открытым исходным кодом и его самостоятельное обслуживание с одновременным обновлением общедоступной версии сопряжено с операционными накладными расходами. Чем больше вы вносите изменений в свою версию, тем больше она расходится с общедоступной версией сообщества. Вы можете посвятить много часов утомительной работы согласованию изменений между версией с открытым исходным кодом и вашей версией. Внесение изменений непосредственно в общедоступную версию помогает снизить затраты на поддержку уникальных изменений, способных нарушить работу вашей инфраструктуры.

13.2 Обновление инструментов

Если вы работаете с IaC в течение нескольких лет, то рано или поздно наступает момент, когда вам нужно обновить инструмент или плагины, которые он использует. Большой разрыв между рабочей версией вашего инструмента и последней версией может значительно затруднить обновление вашей инфраструктуры!

Вы узнали об этой проблеме управления версиями модулей в главе 5. В этом разделе я расскажу о некоторых соображениях и приемах, которые вы можете использовать, когда вам нужно обновить свой инструмент.

ПРИМЕЧАНИЕ Вы не найдете волшебных инструментов, которые идеально сделают все за вас. Обновление инструмента всегда будет иметь серьезные препятствия. Уникальные паттерны в вашей IaC (например, встроенные сценарии) могут привести к сбою системы во время обновления! Поэтому постарайтесь по возможности ограничить сложность логики IaC.

Представьте, что вы проводите аудит инструментов IaC компании (начальное конфигурирование, управление конфигурацией и построение образов). Большая часть IaC компании использует инструмент версии 1.7. Однако последняя версия инструмента – 4.0. Ваш первый крупный проект предполагает обновление IaC для использования инструмента версии 4.0.

13.2.1 План действий перед обновлением

Прежде чем приступить к обновлению инструмента или плагина, ознакомьтесь с несколькими важными практиками, которые помогут свести к минимуму потенциальное нарушение работы вашей инфраструктуры. Ваш план действий должен включать несколько шагов по разделению, стабилизации и согласованию IaC.

Этот план схематически показан на рис. 13.5. Вы разделяете все зависимости, проверяете управление версиями, фиксируете все версии и развертываете IaC, чтобы уменьшить дрейф.



Рис. 13.5 Ваш список действий перед обновлением инструмента должен включать фиксацию и проверку всех версий модулей, плагинов и инструментов

Если при обновлении инструмента добавляются или удаляются поля, вам необходимо передать правильную информацию, ожидаемую каждым подмножеством ресурсов. Внедрение зависимостей образует уровень абстракции для атрибутов конфигурации (см. главу 4). Он защищает каждое подмножество от изменений в другом подмножестве.

Вы проверяете, все ли модули IaC зарегистрированы в системе управления версиями (глава 5). Аналогичным образом вы проверяете, что все IaC или репозитории, использующие вывод модуля, относятся к определенной версии. Например, одна из команд компании, занимающейся социальными сетями, всегда использует последнюю версию модуля. Вы добавляете модуль версии 2.3.1 в репозиторий команды. Когда вы выпускаете версию модуля 3.0.0 с обновленным инструментом, IaC группы не будет обновлена в связи с критическими обновлениями. Вы можете обновить модуль и отправить критические обновления всем, кто его использует, не фиксируя версию модуля.

Вы также убеждаетесь, что каждая команда фиксирует текущую версию инструмента и версии его плагинов. Хотя плагины могут не иметь прямой совместимости, вам следует сохранить текущую версию и избежать добавления новой конфигурации в другую версию. Наконец, вы помещаете все зафиксированные версии модулей, инструментов и плагинов в каждый модуль и конфигурацию IaC. Таким образом, фиксация версий не приведет к появлению новых изменений.

После завершения подготовки к обновлению необходимо спланировать путь обновления инструмента. На рис. 13.6 показано, что обновление инструмента с версии 1.0 до версии 4.0 вносит критические изменения! Вы принимаете решение обновиться до версии 3.0, которая обеспечивает обратную совместимость с версией 1.7. Затем вы можете обновить версию 3.0 до 4.0, что должно уменьшить количество критических изменений за один переход.

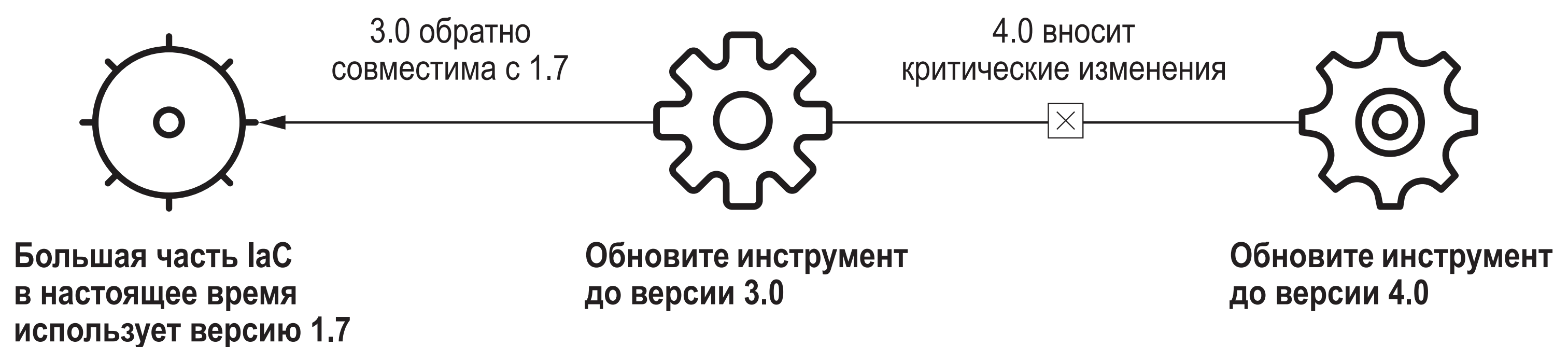


Рис. 13.6 Вы планируете путь обновления инструмента с учетом наличия обратно совместимых версий и версий с критическими изменениями

Не обновляйте инструмент до последней версии сразу. Вместо этого изучите список критических изменений в вашем инструменте и подумайте, сможете ли вы внести эти изменения без ущерба для системы. Я избегаю обновления более, чем на две версии (или под-версии в случае бета-версий) одновременно. Большинство инструментов имеют критические изменения в поведении или синтаксисе.

Постарайтесь проверить обновления в тестовой среде, прежде чем развертывать их в производстве. Наблюдая за своей системой и ее тестовой средой, вы поймете, способно ли обновление инструмента нарушить инфраструктуру. Серьезные обновления никогда не проходят так гладко, как нам хотелось бы, а тестирование помогает выявить серьезные проблемы перед обновлением производственной среды.

13.2.2 Обратная совместимость

Многие инструменты IaC обладают определенной обратной совместимостью версий. Обычно они поддерживают одновременно старые и новые функции в одной или двух версиях, прежде чем окончательно исключить старые функции. Даже если инструмент поддерживает обратную совместимость, обязательно выполните рефакторинг инфраструктуры под новые функции как можно скорее.

В нашем примере инструмент обновляется с версии 1.7 до 3.0. К счастью, версия 3.0 поддерживает обратную совместимость с версией 1.7. Она предлагает новые функции, но не содержит критических изменений, которые могли бы повлиять на работоспособность вашей IaC. Тем не менее любое обновление инструмента требует вашего пристального внимания.

Вы приступаете к обновлению вместе с членами команды новостной ленты, поскольку они согласны помогать вам во время обнов-

ления. Команда развертывает все изменения и следит за тем, чтобы с этого момента в IaC не вносились никакие обновления. Затем вы изучаете актуальную конфигурацию, чтобы найти лучший способ обновить инструмент, не нарушая работу социальных сетей.

На рис. 13.7 показано применение метода рефакторинга из главы 10 для обновления инструмента. Вы начинаете с ресурсов высокого уровня, поскольку другие ресурсы от них не зависят, развертываете изменения, тестируете систему и обновляете ресурсы более низкого уровня.

Вы уведомляете команду новостной ленты, что сначала обновите ресурсы инфраструктуры DNS и балансировщик нагрузки. Остальные ресурсы от них не зависят. Обновление этих ресурсов позволит вам проверить, работают ли ваши методы обновления. Обновите версию инструмента для ресурса и запустите IaC. Проверьте результаты пробного прогона и тестирования в своем конвейере поставки, чтобы убедиться, что вы ничего не нарушили.

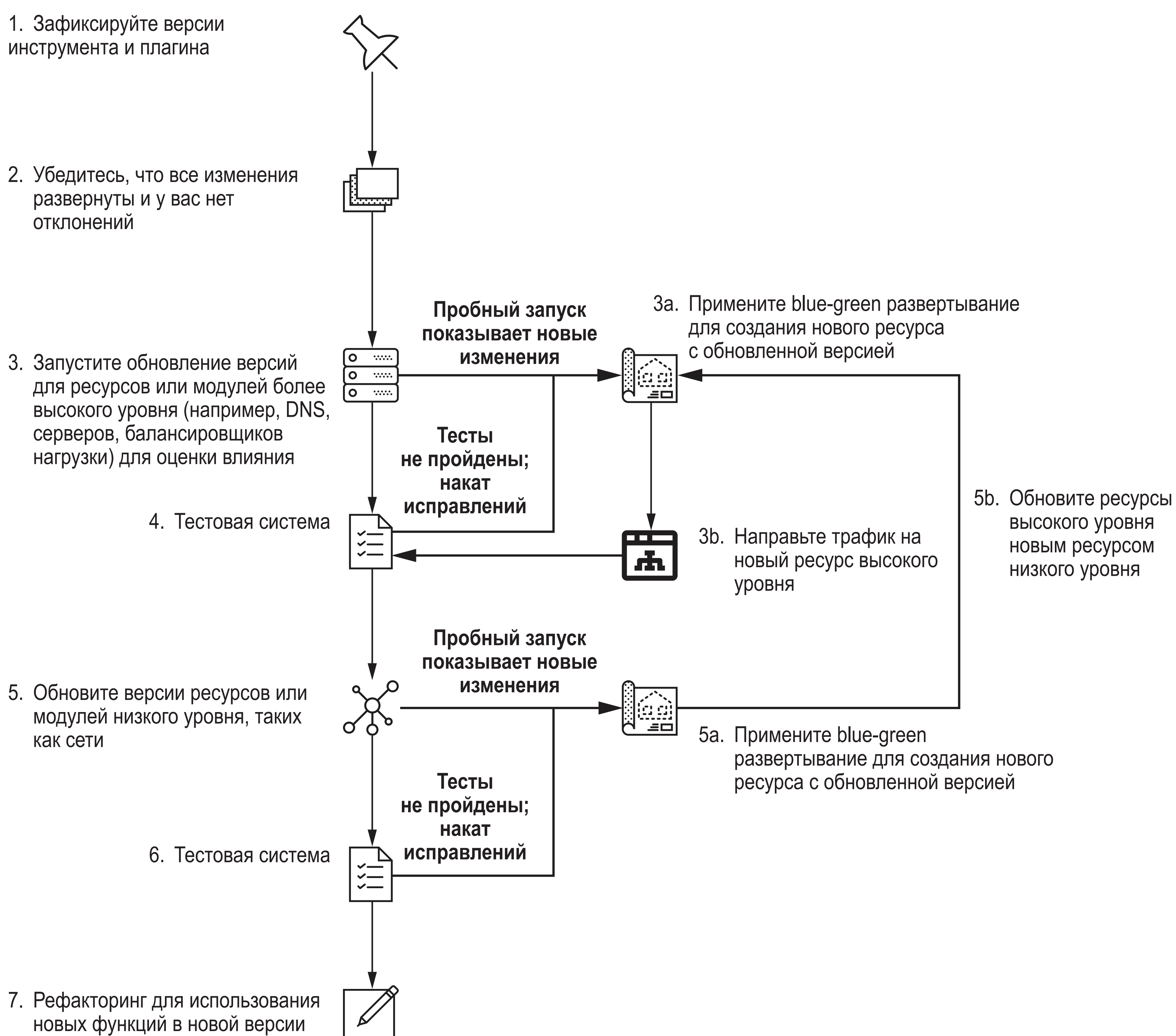


Рис. 13.7 Применение методов рефакторинга для последовательного обновления обратно совместимой версии инструмента начиная с ресурсов высокого уровня

Допустим, ресурсы высокого уровня, такие как DNS и балансировщик нагрузки, обновились без каких-либо сбоев. Далее вы переходите к ресурсам более низкого уровня. Вы начинаете обновлять серверы по методу скользящего обновления из главы 10. Вместо того чтобы обновлять все серверы одновременно, вы начинаете с одного из них и сразу сталкиваетесь с проблемой.

В конфигурации производственного сервера имеется сценарий переопределения, который перестает работать при обновлении инструмента. К счастью, вы затронули только один сервер, поскольку использовали скользящее обновление. Ведь вы старались сделать так, чтобы лента социальных сетей всегда работала и была доступна.

На рис. 13.8 показано применение метода повтора транзакций из главы 11, чтобы исправить сервер, который перестал работать. Вы выполняете исправление вручную и отлаживаете исходный сервер для устранения проблемы. Устранив проблему в тестовой среде, вы переносите исправление на производственный сервер и продолжаете скользящее обновление серверов.

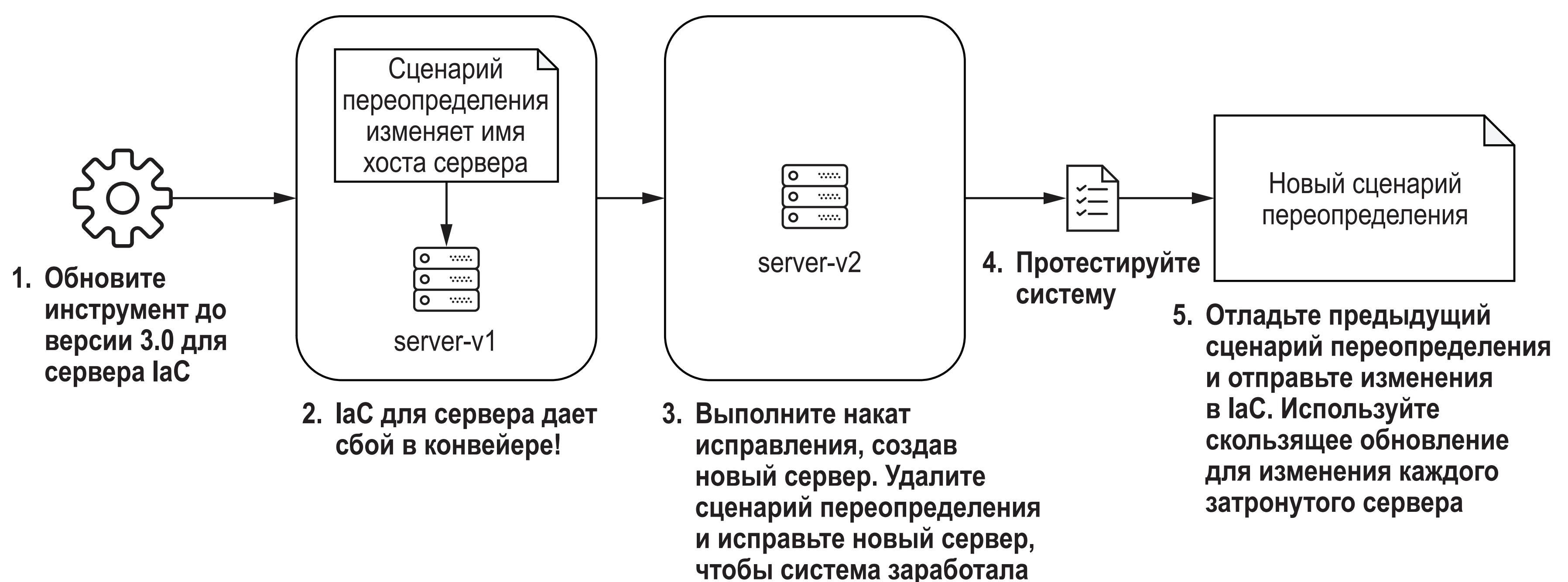


Рис. 13.8 Воспользуйтесь методом скользящего обновления, чтобы минимизировать ущерб при неудачном обновлении инструмента и выполнить накат к исправному состоянию, если изменение не удалось

Серверы успешно прошли тестирование. Вы переходите к ресурсу самого низкого уровня – сети. На всякий случай вы развертываете вторую версию сети в соответствии с методом blue-green развертывания из главы 9. После развертывания новой сети вы запускаете все сквозные тесты и проверяете, что система по-прежнему работает. Вы завершили обновление инструмента! Зачем мы вспомнили про такие методы, как рефакторинг, blue-green развертывание и скользящие обновления? Вы должны на каждом шаге стараться минимизировать ущерб от потенциального сбоя. Эти методы могут показаться избыточными, но они предлагают структурированный и менее рискованный подход к обновлению вашей системы. Изменение инструмента связано с изменением инфраструктуры, поэтому мы применяем очень похожие методы с аналогичным результатом.

Я рекомендую использовать метод скользящих обновлений при обновлении IaC серверов или других вычислительных ресурсов. blue-green развертывание помогает обновлять инструменты для ресурсов инфраструктуры нижнего уровня с высоким уровнем риска. Ресурсы верхнего уровня обычно удается обновить «на ходу».

Однако вы не можете полностью исключить вероятность сбоя. В случае сбоя системы используйте методы и паттерны повтора транзакций из главы 11. Ресурсы высокого уровня можно вернуть к работе простым восстановлением конфигурации, тогда как ресурсы более низкого уровня проще создать заново вместе с пакетом предыдущих обновлений.

13.2.3 Критические изменения в новых версиях

Периодически вы будете обнаруживать, что ваш инфраструктурный инструмент или плагин вышел в новой версии с критическими изменениями. Новые версии с критическими изменениями часто встречаются на начальном этапе разработки инструмента, например когда инструмент получает новые варианты применения или функции. Если вам необходимо перейти на версию инструмента с критическими обновлениями, воспользуйтесь методами внесения изменений, описанными в главе 9.

По просьбе компании, занимающейся социальными сетями, мы обновили инструмент ленты новостей с версии 1.7 до 3.0. Однако обновление с версии 3.0 до 4.0 включает в себя критические изменения! Версия 4.0 содержит изменения в схеме серверной части, которые могут повлиять на другие ресурсы. Как обновить инфраструктуру до версии 4.0, не влияя на систему?

В главе 4 я упоминала о существовании *инструментального состояния*. Инструменты хранят копию состояния инфраструктуры, чтобы обнаружить расхождения между фактическим состоянием ресурса и конфигурацией, а также отслеживать, какими ресурсами он управляет. Инструментальное состояние отличается от фактического состояния инфраструктуры. При обновлении инструмента необходимо отделить старое инструментальное состояние от нового.

В нашем примере необходимо изолировать старое состояние версии 3.0 от нового состояния версии 4.0. Разделение инструментального состояния минимизирует потенциальный ущерб от сбоя за счет изоляции ресурсов инфраструктуры, которые инструмент должен изменить. Чем меньше ресурсов затронуто, тем быстрее восстановление и меньше влияние на другие части системы.

На рис. 13.9 каждая команда в компании, занимающейся социальными сетями, размещает инструментальное состояние в разных местах. Разделение инструментальных состояний гарантирует, что изменения в синей инфраструктуре команды новостной ленты не повлияют на зеленую инфраструктуру.

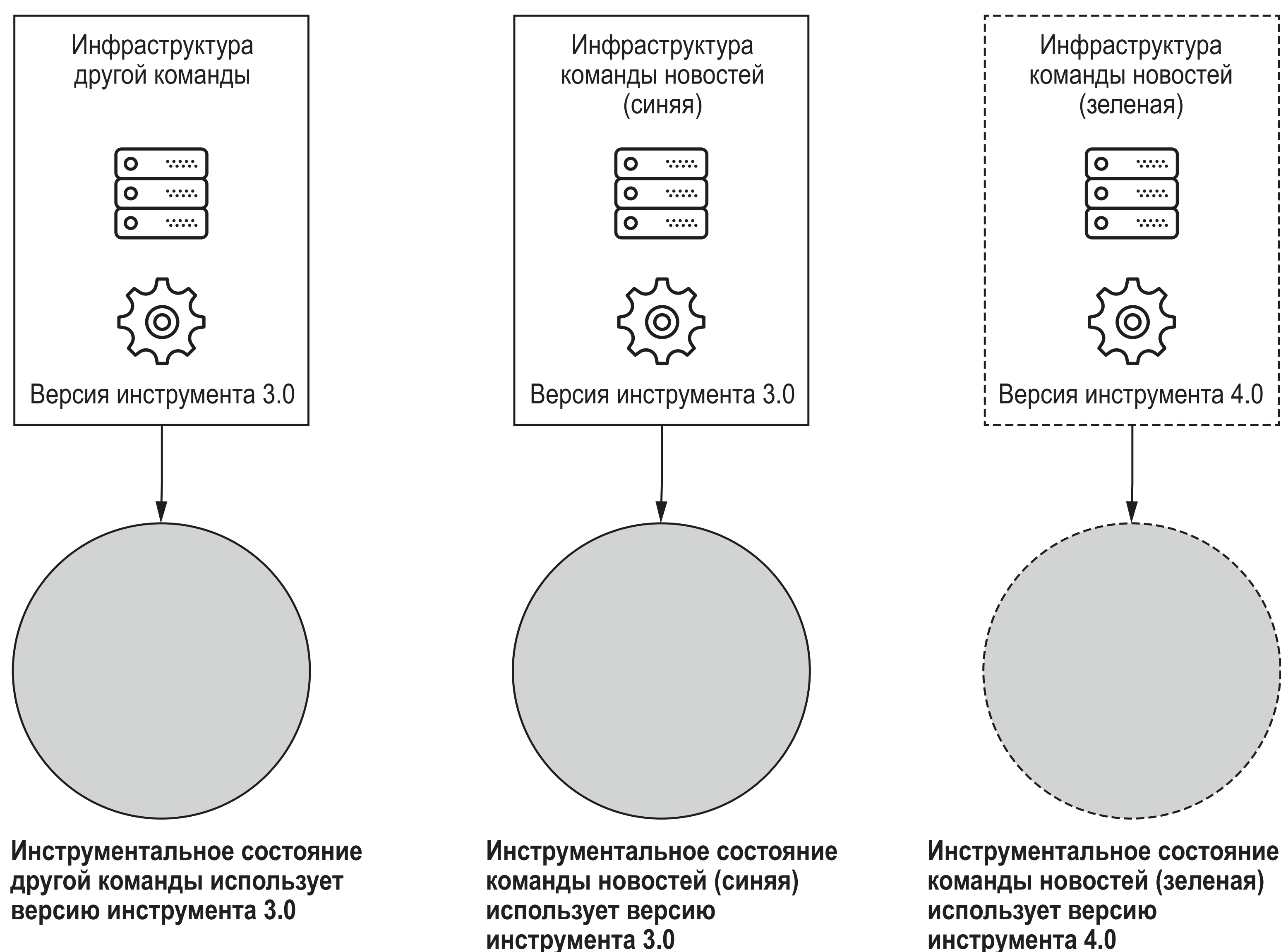


Рис. 13.9 Инструментальное состояние отражает состояние инфраструктуры, позволяющее инструменту проводить сравнения, и может быть разделено по разным местам

Сначала вы фиксируете версии инструментов и модулей в IaC команды новостной ленты. Затем обновляете модули инфраструктуры до версии 4.0. Вы выпускаете новую версию модуля, отмечая критические изменения.

Далее копируете существующую конфигурацию в новый каталог. Каждый новый каталог создает новое инструментальное состояние. Найдите каталог сети и создайте новую сеть с помощью инструмента версии 4.0. Теперь у вас должны быть исходные «синие» ресурсы, созданные при помощи версии 3.0, и новые «зеленые» ресурсы, созданные при помощи версии 4.0.

Вы применили *blue-green стратегию развертывания к инструментальному состоянию*, чтобы создать новые ресурсы с помощью нового инструмента. Создание нового набора ресурсов с новой версией гарантирует, что любые критические изменения не повлияют на существующую инфраструктуру.

ОПРЕДЕЛЕНИЕ *Blue-green развертывание инструментального состояния* – это метод, в соответствии с которым создают новое подмножество ресурсов инфраструктуры с помощью новой версии инструмента. Затем вы постепенно перемещаете трафик со старого (синего) набора ресурсов на новый (зеленый) набор. Метод изолирует критические изменения в новом наборе ресурсов для тестирования.

После создания ресурсов низкого уровня скопируйте ресурсы высокого уровня в новый каталог. Обновите его зависимости, чтобы использовать ресурсы низкого уровня из инструмента версии 4.0. В итоге вы должны добиться того, чтобы все ресурсы использовали новую версию инструмента.

Эта стратегия показана в общем виде на рис. 13.10. Создайте ресурсы высокого уровня, выполните тесты и отправьте трафик на новые ресурсы.

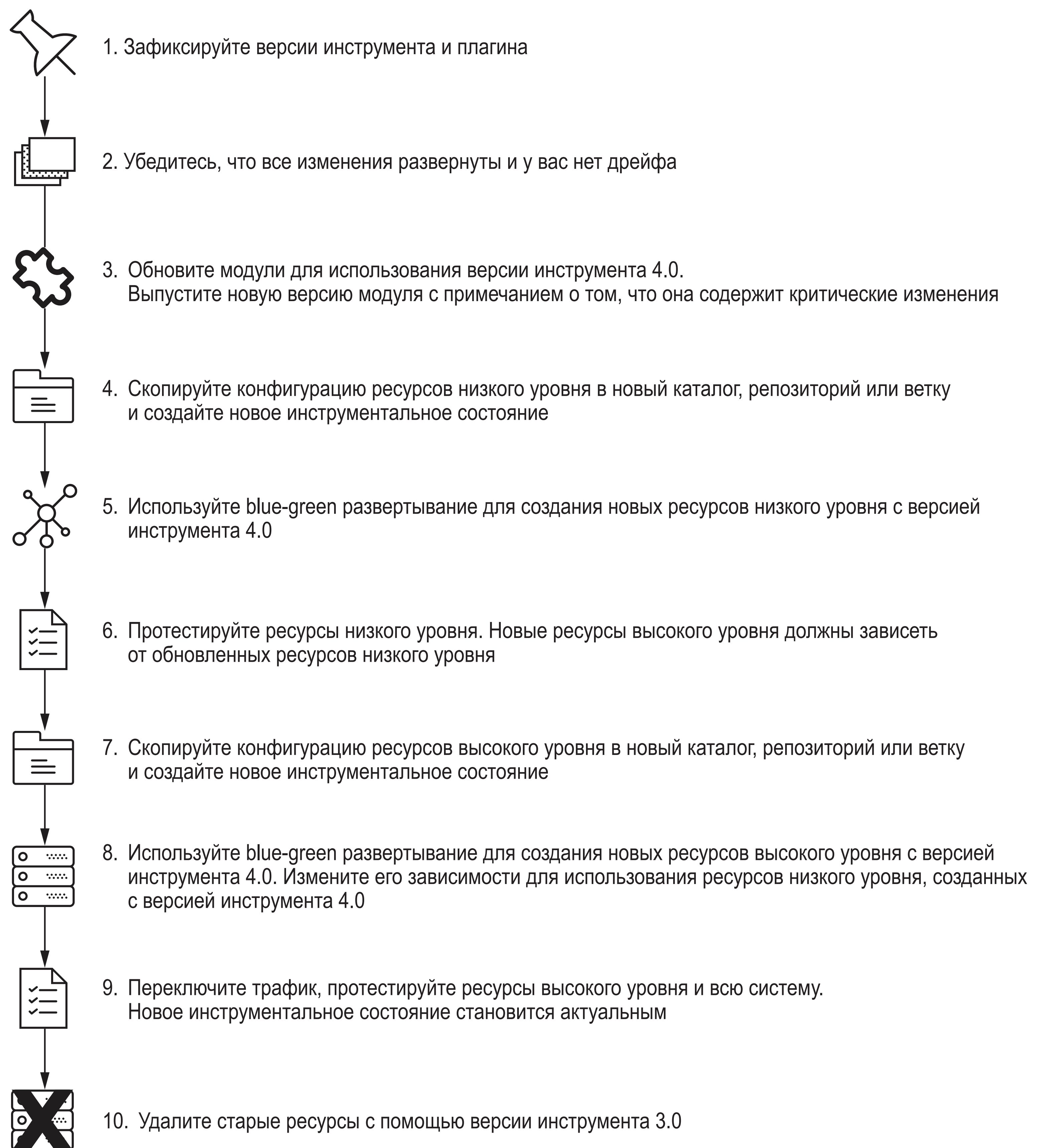


Рис. 13.10 При наличии критических изменений рекомендуется создать новый ресурс с другим инструментальным состоянием с помощью последней версии инструмента и переключить на него трафик после тестирования

Этот подход отличается от blue-green развертывания, описанного в главе 9. Вы создаете новый набор ресурсов и совершенно новое инструментальное состояние. Если в вашей инфраструктуре ресур-

сы связаны достаточно слабо, подмножества ресурсов могут иметь разные версии инструментов, не влияя на функциональность системы.

Как вы помните, работа с IaC бывает основана на разных структурах репозитория (глава 5) и моделях ветвления (глава 7). В зависимости от структуры вашего репозитория и модели ветвления вы можете по-разному изолировать состояние. Вы всегда можете выделить новую ветку и изменить ее конвейер для развертывания инфраструктуры или скопировать отдельный каталог в основную конфигурацию репозитория.

Предположим, ваша организация придерживается строгого подхода к конвейерам поставки и разрешает развертывание производства только в ветке `main`. В этом случае вы можете создать новый репозиторий для обновления инструмента и заархивировать старый.

Если у вас есть возможность провести пробный прогон и тщательно протестировать изменения, то имеет смысл применить изменение «на ходу». Если это не удалось, создайте с помощью новой версии инструмента совершенно новый ресурс с новым инструментальным состоянием. В случае сомнений обратитесь к документации по обновлению инструмента. Как правило, я пробую выполнить обновление «на ходу», если инструмент предлагает какой-либо способ или сценарий миграции, упрощающий обновление.

13.3 Замена инструментов

Я постаралась сделать эту книгу как можно более независимой от инструментов, предлагая при этом конкретные примеры, демонстрирующие паттерны и методы. Тем не менее я понимаю, что когда-то мне придется обновить эту книгу и рассказать о более новых и лучших технологиях! При использовании IaC в течение некоторого времени вы неизбежно меняете инструменты в погоне за лучшей функциональностью и поддержкой провайдеров. Какие приемы следует использовать при переходе на новый инструмент?

Многие подходы, описанные в этой книге, помогут защитить вашу систему от негативных последствий обновлений. Определение области действия и реализация модульности вашей инфраструктуры с помощью паттернов из главы 3 и разделение зависимостей с помощью паттернов из главы 4 помогают командам применять инструменты, соответствующие бизнес-сценариям, и заменять их по мере необходимости. Если вы незнакомы с такими подходами, то рискуете столкнуться с трудностями при переходе на новый инструмент.

Представьте, что вы завершаете обновление инструментов IaC для компании, занимающейся социальными сетями. Вы уже предвкушаете перерыв на кофе, когда члены сетевой команды просят вас

о помощи. Они хотят перейти от специфического DSL провайдера к DSL с открытым исходным кодом. Их конфигурация потребует дополнительной проверки со стороны команды новостной ленты, которая ничего не знает о провайдере и его коде.

В ходе вашего исследования не удалось найти решения или инструменты с открытым исходным кодом, которые могли бы облегчить переход. Было бы удобно иметь хоть что-то, помогающее перевести конфигурацию с проприетарного DSL провайдера на DSL с открытым исходным кодом. Без этого вам и сетевой команде придется очень осторожно выполнять переход вручную.

13.3.1 Если новый инструмент поддерживает импорт

Допустим, вы не смогли найти автоматизированное решение для перехода между инструментами. Однако вы можете применять паттерны, описанные в этой книге. Некоторые инструменты поддерживают возможность импорта для добавления новых ресурсов в инструментальное состояние. Вы можете использовать методы, описанные в главе 2, для переноса существующих ресурсов в новый инструмент.

На рис. 13.11 показано обновление модулей для использования нового DSL с открытым исходным кодом. Вы также обновляете соответствующие тесты. Для начала определите некоторые низкоуровневые ресурсы, которые вы можете изменить. Создайте отдельный каталог, ветку или репозиторий, чтобы изолировать свой новый DSL от DSL провайдера. После написания конфигурации на новом DSL вы импортируете существующие ресурсы в состояние нового DSL.

Вы переписываете тесты, чтобы протестировать новый синтаксис конфигурации на DSL с открытым исходным кодом. Тесты готовы, и вы переходите к написанию конфигурации и импорту высокоуровневых ресурсов. Наконец, вы удаляете файл IaC.

На протяжении каждого цикла написания новой IaC с использованием DSL с открытым исходным кодом вам необходимо выполнять пробные запуски и при необходимости переписывать тесты. Пробный прогон показывает, соответствуют ли значения по умолчанию для нового инструмента существующему состоянию. В случае несоответствия необходимо изменить новую IaC и устранить дрейф.

13.3.2 Если в новом инструменте нет функции импорта

Некоторые инструменты не поддерживают возможности импорта, и вам придется вручную создать ресурсы для нового инструмента. Представьте, что сетевая команда попросила вас помочь им перейти с DSL одного провайдера на другого. Однако DSL нового провайдера не позволяет импортировать существующие ресурсы в свое состояние.

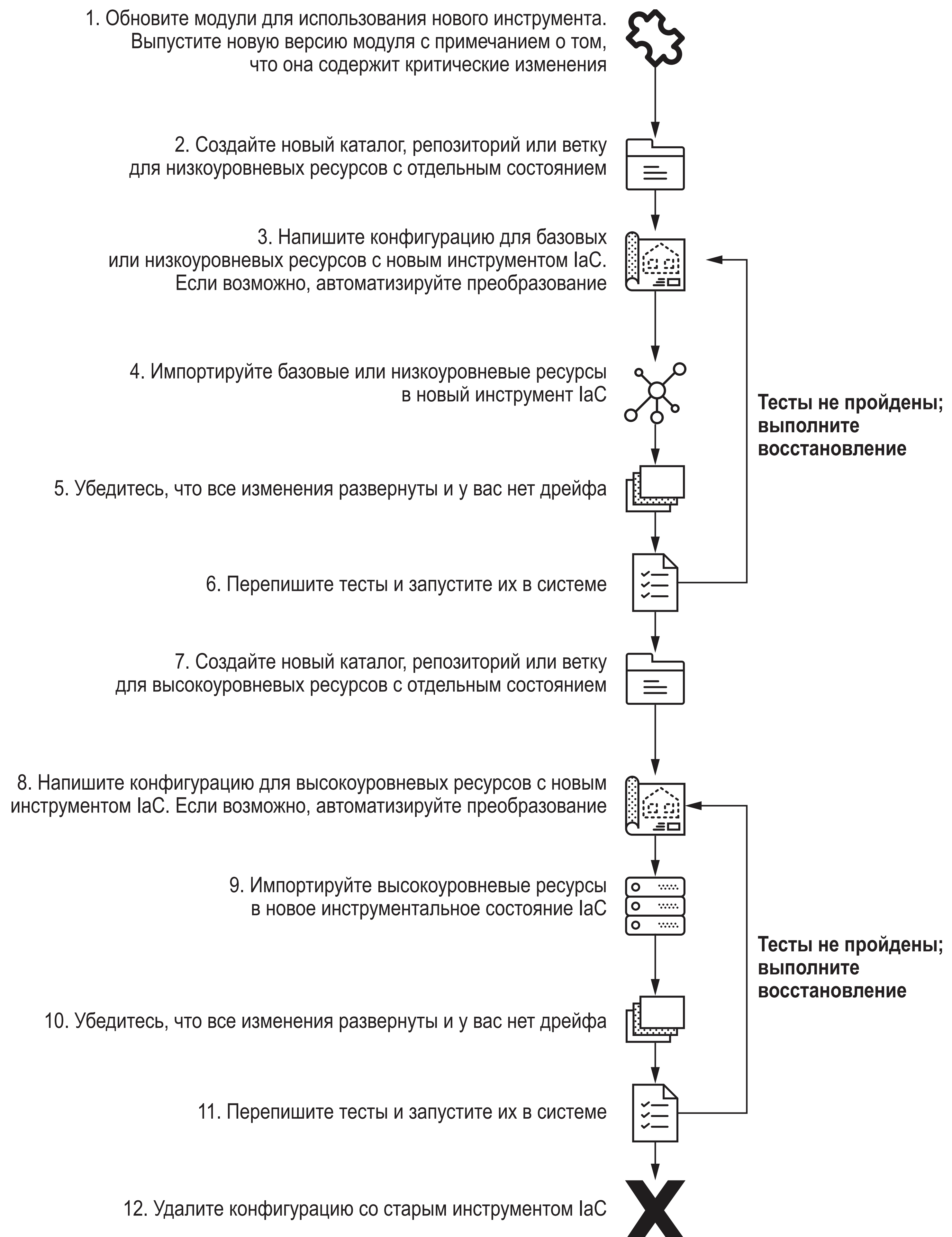


Рис. 13.11 Новый инструмент, поддерживающий импорт ресурсов, позволяет перейти без изменения существующих ресурсов

Если ваш новый инструмент не поддерживает импорт существующих ресурсов, вам необходимо воссоздать ресурсы в новом инструментальном состоянии, придерживаясь стратегии blue-green развертывания. В соответствии со схемой на рис. 13.12 вы начинаете с написания новой IaC для ресурсов низкого уровня и тестов рефакторинга для нового инструмента. Через несколько итераций вы завершаете перенос ресурсов высокого уровня, переключаете трафик и тестируете систему целиком.

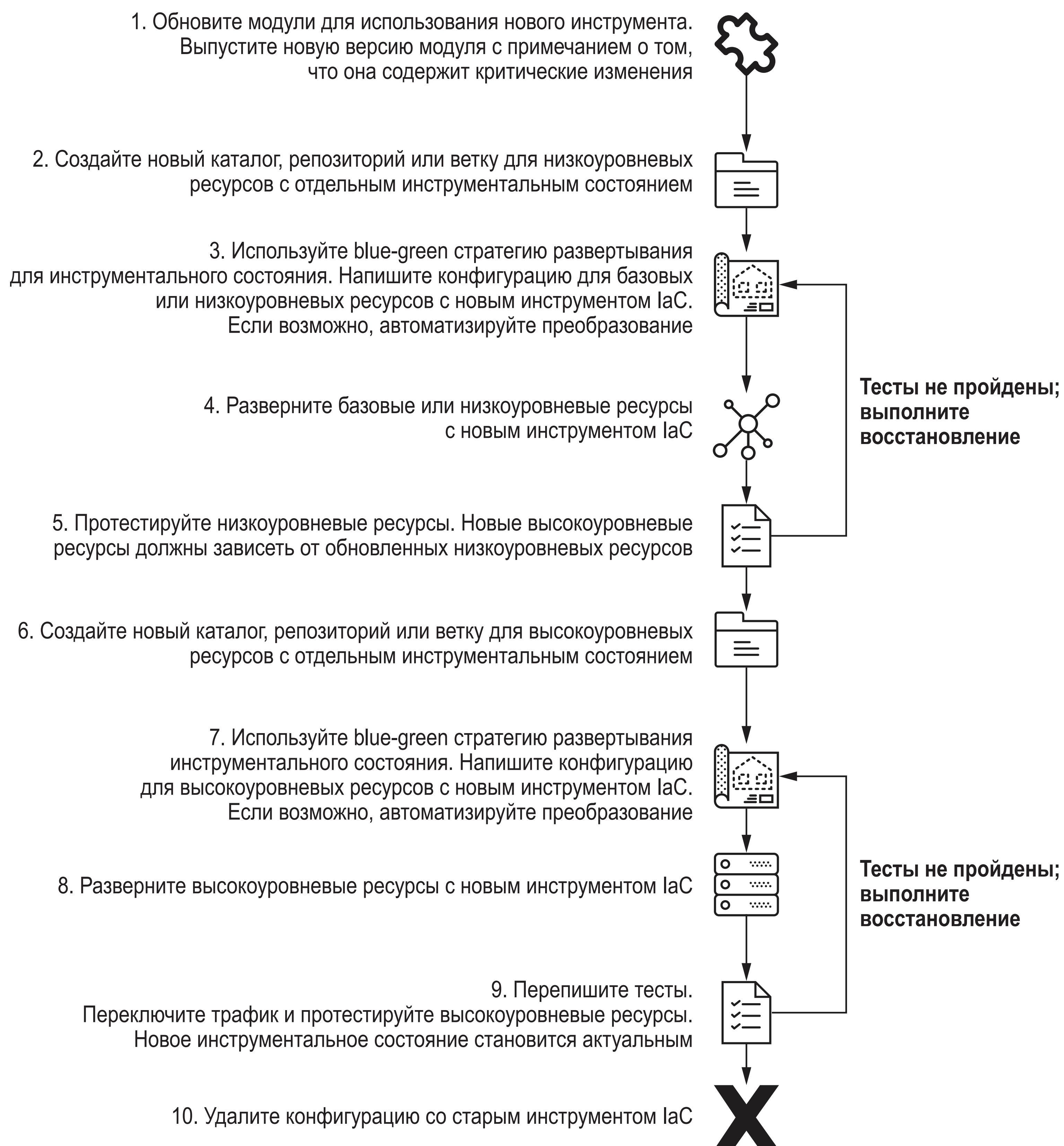


Рис. 13.12 Новый инструмент, который не может импортировать ресурсы, требует перехода согласно стратегии blue-green развертывания

Схема перехода на новый инструмент в целом остается неизменной, независимо от наличия функции импорта. Однако переход без возможности импорта требует больше усилий, поскольку вы заново создаете систему. Даже если у вас есть волшебный скрипт для перехода с одного инструмента на другой, все равно полезно знать паттерны и методы и уметь их применять, чтобы избежать повреждения критически важных ресурсов инфраструктуры, таких как сеть.

Вы всегда будете добавлять, заменять или обновлять инструменты в своей инфраструктурной экосистеме. Ваша организация выберет инструменты, соответствующие ее архитектурным приоритетам. Развитие IaC будет зависеть от ваших навыков применения методов модульности, изоляции и управления IaC. Я всегда держу в памяти паттерны и методы, чтобы в нужный момент

внести изменения в инфраструктуру, IaC, модули, инструменты и организационную структуру и снизить риски для критически важных систем.

Независимо от того, имеет ли инструмент возможности импорта, вам необходимо переписать тесты для каждого ресурса, который вы пересоздаете. На рис. 13.13 показано, что при миграции необходимо провести рефакторинг модульных и контрактных тестов для каждого модуля и подмножества ресурсов. Однако ваши сквозные и интеграционные тесты могут остаться прежними.

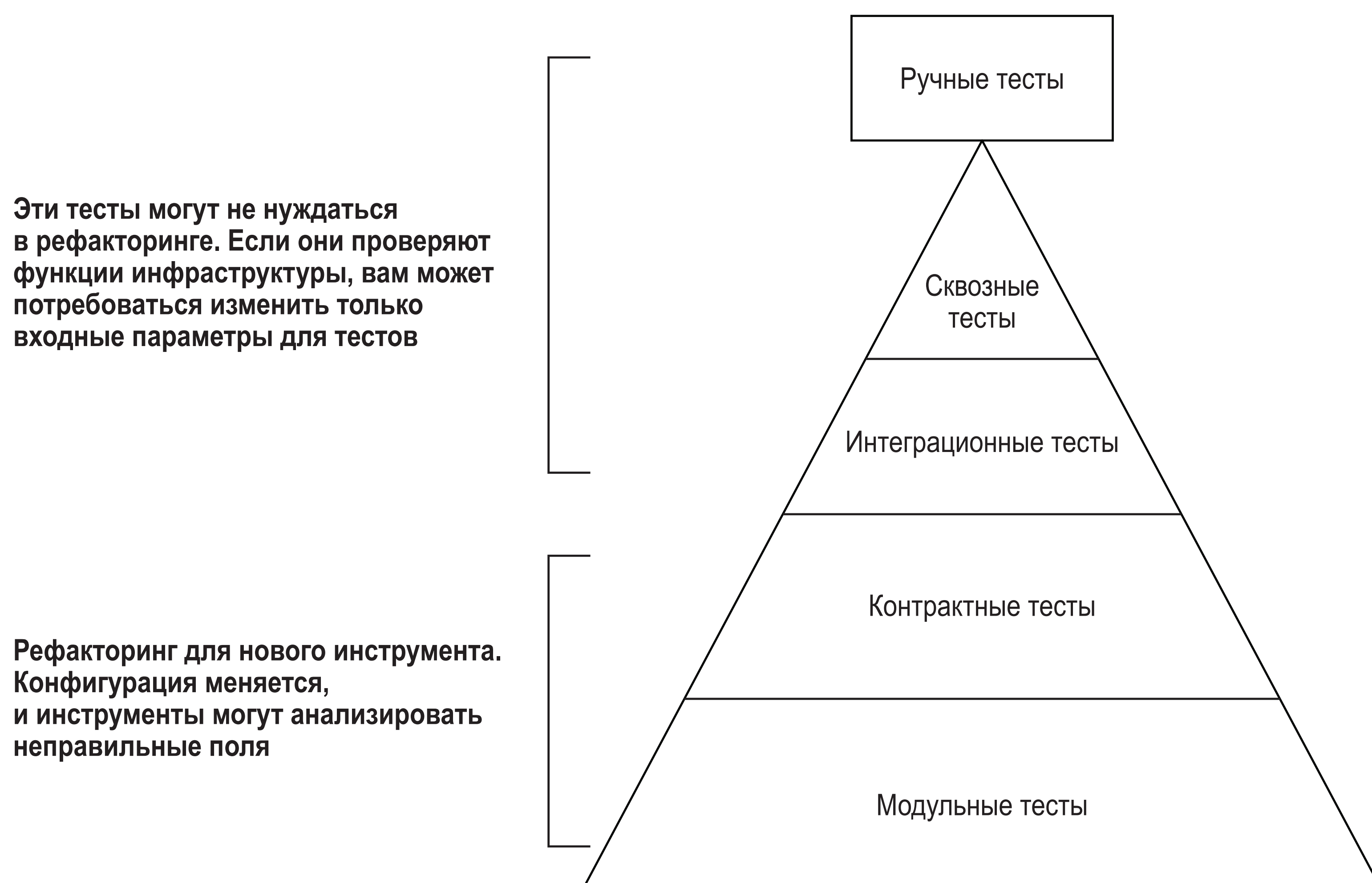


Рис. 13.13 При обновлении инструмента вам придется переписать модульные и контрактные тесты, тогда как интеграционные, сквозные и ручные тесты преимущественно могут остаться прежними

Новый инструмент повлияет на модульные и контрактные тесты, поскольку он использует другой формат состояния и метаданных. Тесты не могут проанализировать правильную информацию из нового инструмента. Интеграционные и сквозные тесты, скорее всего, останутся прежними, так как они оценивают функции инфраструктуры, а не самого инструмента.

Рефакторинг тестов по ходу работы заключается в добавлении дополнительных тестов, удалении избыточных тестов и внесении обновлений в более широкие тесты безопасности или политики. Вам следует обновить входные параметры своих ручных, интеграционных и сквозных тестов, потому что у вас появились новые ресурсы. Однако сами тесты не сильно меняются, поскольку они проверяют функциональность системы, а не атрибуты инфраструктуры.

13.4 IaC, управляемая событиями

Большая часть этой книги посвящена методам совместной работы и разработки IaC, направленным на уменьшение последствий сбоя критически важных систем. Ознакомившись с ключевыми принципами, вы сможете распространить их на более динамичные варианты реализации IaC.

Например, команде разработчиков нужна очень динамичная автоматизация IaC. Всякий раз, когда члены команды развертывают новый экземпляр приложения, им необходимо обновить правило брандмауэра, чтобы разрешить доступ экземпляра к базе данных. Вместо того чтобы разворачивать новый экземпляр, а затем не забывая обновлять правило брандмауэра, они хотят сделать так, чтобы скрипт автоматизации запускал модуль инфраструктуры для настройки правила брандмауэра после запуска экземпляра приложения.

На рис. 13.14 показана схема необходимой автоматизации. Вы развертываете новое приложение с новым IP-адресом. Скрипт автоматизации обнаруживает новый IP-адрес и запускает IaC. Конфигурация обновляет брандмауэр новым IP-адресом. Этот скрипт срабатывает каждый раз, когда вы развертываете новое приложение с новым IP-адресом.



Рис. 13.14 Каждый раз, когда экземпляр приложения получает новый IP-адрес, модуль инфраструктуры обновляет правило брандмауэра, добавляя в него этот адрес

Вы можете предусмотреть возможность автоматического запуска модуля инфраструктуры при изменении системы. Зависимость от событий в случае IaC означает запуск модуля с минимальной об-

ластью воздействия для настройки инфраструктуры в ответ на событие. Автоматизацию по событию можно применять для обновления другого ресурса или исправления системы.

ОПРЕДЕЛЕНИЕ *IaC, управляемая событиями* (event-driven IaC), запускает модуль инфраструктуры с минимальной областью действия для настройки инфраструктуры в ответ на событие.

Обновление приложения приравнивается к событию. Какой-то сценарий, приложение или система автоматизации обнаруживает событие и реагирует запуском инфраструктурного модуля. Вы можете написать свой собственный скрипт автоматизации или найти инструмент с открытым исходным кодом, чтобы обнаруживать событие и реагировать на него. К средствам автоматизации, которые вы можете использовать для обнаружения событий и реагирования на них, относятся операторы Kubernetes, бессерверные функции или приложения, которые получают данные из очереди событий.

Разве это не GitOps?

В главе 7 я упоминала, что ряд практик и паттернов в этой книге очень похожи на GitOps. Методика GitOps сочетает в себе декларативную конфигурацию, обнаружение дрейфа, контроль версий и непрерывное развертывание. Этот подход идеально соответствует потребностям IaC, управляемой событиями. Я считаю GitOps подмножеством IaC, управляемой событиями, поскольку его автоматизация реагирует на события в изменении конфигурации.

Если фреймворк GitOps обнаруживает дрейф, он запускает сценарий для согласования конфигурации. Например, оркестратор контейнеров Kubernetes использует контроллеры для автоматического согласования декларативной конфигурации с состоянием ресурса. Но автоматизация IaC охватывает более широкий спектр событий, а не только дрейф.

IaC, управляемая событиями, наиболее широко применяется в динамических сервисах и приложениях. Если вы используете IaC, управляемую событиями, старайтесь придерживаться следующих правил:

- избегайте ресурсов инфраструктуры, создание или настройка которых занимает много времени. Никому не понравится ресурс инфраструктуры, на создание которого уходит час;
- не добавляйте слишком много ресурсов в модуль, управляемый событиями. В противном случае вам потребуется много времени на создание большого количества экземпляров;
- найдите баланс между временем, необходимым для внесения изменений в модуль, и интервалом между событиями;

- объедините подходы модульной инфраструктуры из главы 2 с паттернами тестирования из главы 6, чтобы убедиться, что управляемая событиями IaC работает быстро и правильно.

Некоторые события происходят с большой частотой. Вам нужна инфраструктура, которая разворачивается быстрее, чем случается следующее событие, или сценарий автоматизации для пакетной обработки накопившихся изменений через определенный интервал. Выберите минимальное подмножество ресурсов инфраструктуры для развертывания с помощью IaC, управляемого событиями.

На пути от простой статической IaC, которую вы развертываете на основе коммитов кода, до динамической IaC, управляемой событиями, вы будете применять одни и те же паттерны, практики и принципы управления и совместной работы. Ваши цели, потребности вашей команды и бизнес вашей организации будут развиваться и меняться с течением времени – надеюсь, что ваш опыт работы с IaC будет расти вместе с ними. Помните о стратегии тестирования, стоимости инфраструктуры, а также о ее безопасности и соответствии нормативным требованиям.

Краткое содержание главы

- Проверьте функциональность, безопасность и жизненный цикл инструмента или модуля с открытым исходным кодом, прежде чем внедрять его в своей организации.
- Инструмент или модуль с открытым исходным кодом должен иметь значения или поведение по умолчанию, обеспечивающие предсказуемость и стабильность изменений. В противном случае вам придется написать свой код, чтобы добавить атрибуты по умолчанию, принятые в вашей организации.
- Защитите свою инфраструктуру от атак на цепочку поставки, сканируя инфраструктурный инструмент или модуль на предмет безопасности, проверяя сбор сторонних данных и проводя тесты безопасности и комплаенса.
- Форма сопровождения модуля с открытым исходным кодом, а также лицензия, связанная с инструментом или модулем, тоже влияют на возможность использования в вашей организации.
- Инструменты и модули с открытым исходным кодом могут иметь лицензию, относящуюся к одной из двух основных категорий: разрешительная и copyleft (антикопирайт).
- Разрешительная лицензия позволяет вам изменять и обновлять модуль или инструмент, при условии что вы включаете в код копию лицензии и оригинальное уведомление об авторских правах.
- Лицензия типа copyleft позволяет вам изменять и обновлять модуль или инструмент, при условии что вы включаете копию ли-

цензии, оригинальное уведомление об авторских правах и обязательно публикуете свою копию с открытым исходным кодом.

- Перед обновлением инструмента IaC разделите зависимости инфраструктуры и зафиксируйте версии модуля, плагина и инструмента.
- Запустите обновление инструмента с обратной совместимостью, проводя рефакторинг ресурсов высокого уровня и постепенно переходя к ресурсам низкого уровня.
- Blue-green стратегия развертывания инструментального состояния означает создание нового набора ресурсов инфраструктуры с состоянием, отличным от существующей конфигурации.
- Инструментальное состояние – это копия состояния инфраструктуры, которую инструмент IaC использует для обнаружения дрейфа или ресурсов, находящихся под его управлением.
- Для обновления инструмента, имеющего критические изменения, применяйте blue-green стратегию развертывания инструментального состояния, начиная с ресурсов низкого уровня и заканчивая ресурсами высокого уровня.
- При замене одного инструмента другим используйте функцию импорта нового инструмента для переноса существующих ресурсов в область охвата нового инструмента. Начните с ресурсов низкого уровня и постепенно перейдите к ресурсам высокого уровня. Затем удалите конфигурацию старого инструмента.
- Если новый инструмент не имеет функции импорта существующих ресурсов инфраструктуры, вам потребуется применить blue-green стратегию развертывания инструментального состояния.
- IaC, управляемая событиями, запускает минимальный модуль IaC в ответ на системное событие для автоматизации изменений инфраструктуры.
- Обеспечьте быстрое развертывание модуля IaC, управляемого событиями, за счет охвата как можно меньшего количества ресурсов.

Дополнительная глава от T1 Облако. О нашем опыте работы с IaC

Уважаемые читатели, команда T1 Облако благодарит вас за внимание, которое вы уделили этой книге. В ней говорится о технологии *инфраструктура как код* (infrastructure as code, IaC) и приводятся примеры создания инфраструктуры, в которых используются Python и HashiCorp Terraform, а в качестве облачного провайдера — Google Cloud Platform. Наверняка у вас возникли вопросы о том, как применить полученные знания при работе с российским облачным провайдером.

14.1 О Terraform-провайдере T1 Облако

Чтобы заказчикам было удобно управлять инфраструктурой, размещенной в нашем облаке, мы разработали свой Terraform-провайдер. Он находится в нашем публичном GitLab-репозитории, его код открыт, и если вы захотите стать нашим контрибьютором, мы будем только рады.

Зайти в репозиторий можно по ссылке <https://clck.ru/3FBPy7>. Там вы найдете подробные инструкции по использованию Terraform,

а также примеры конфигурационных файлов. Эта информация помогает пользователям минимизировать время на обучение и настройку, чтобы быстрее приступить к работе с Terraform. В репозитории также содержится информация о последней версии Terraform-провайдера и поддерживаемых ресурсах, что позволяет командам и клиентам оставаться в курсе последних изменений и улучшений.

14.2 О сценариях использования Terraform-провайдера у клиентов

Расскажем о нескольких сценариях использования Terraform-провайдера T1 Облако.

Например, в 2024 году мы выступали технологическим партнером крупного хакатона и предоставляли участникам современную облачную инфраструктуру. Для каждой из 25 команд нам нужно было поднять окружение – виртуальные машины (ВМ) и масштабируемые хранилища S3. Каждой команде было нужно по 5 ВМ, которые должны были быть связаны между собой. С помощью Terraform-провайдера мы автоматизировали эту работу и развернули необходимое окружение за минимальное время.

Вот еще один пример. Образовательный онлайн-проект запустил виртуальные классы для проведения занятий. На время урока организаторы разворачивали виртуальный класс на 30 типовых ВМ. После окончания урока они удаляли эти ВМ. При этом команда производила изменения в конфигурации инфраструктуры не через веб-консоль облачного провайдера, а с помощью Terraform. Оплата шла по модели pay-as-you-go, то есть за те ресурсы, что используются виртуальными машинами в конкретный момент времени.

Для одного из клиентов мы создали частную инсталляцию облачной платформы, используя инфраструктуру публичного облака. Для настройки данной инфраструктуры применяли Terraform. В итоге получили частную инсталляцию облака за минимальное время и с минимальным использованием ресурсов клиента. Как говорят наши инженеры, «развернули облако в облаке».

В еще одном крупном облачном проекте клиенту требовалось создать группу безопасности со 100 правилами. На каждый порт ему нужно было по одному правилу безопасности, и создавал он их через Terraform. Конечно, это оказалось быстрее и проще, чем если бы он все делал вручную. По оценкам наших разработчиков, для выполнения данной задачи через веб-интерфейс ему пришлось бы совершить 400 кликов. В этом случае Terraform помог минимизировать ошибки, которые могли бы появиться при однотипном вводе информации вручную.

14.3 О развитии Terraform-провайдера T1 Облако

Говоря о наших наработках, важно сказать, что вклад в развитие Terraform-провайдера T1 Облако вносим не только мы, но и наши клиенты. Например, среди них есть компании, которые хотят стать или уже стали нашим контрибьютором. Они заинтересованы в доработке провайдера для учёта своих специфических требований, добавлении новых ресурсов, а также устранении багов. Клиенты могут оставлять свои предложения по доработке кода в заявках в нашем репозитории. Это способствует повышению качества оказываемых нами услуг.

14.4 Цифры и прогнозы

По данным различных исследований о состоянии DevOps в России, Terraform используют около 37 % специалистов. Со своей стороны мы наблюдаем динамику повышения спроса на этот инструмент среди клиентов на 40 %. По нашим прогнозам, в ближайшие три года использование Terraform в России будет демонстрировать устойчивый рост. Это обусловлено несколькими причинами: увеличением интереса к подходу IaC, автоматизации процессов DevOps, необходимостью оптимизировать управления облачной инфраструктурой и снижать затраты на ее обслуживание.

14.5 Примеры кода

Примеры кода играют ключевую роль в понимании и освоении Terraform. Ниже мы показали три примера, которые позволят читателям лучше понять, как Terraform-провайдер помогает автоматизировать развертывание и управление ресурсами в облачной среде. Приведенные фрагменты кода вы можете адаптировать в своих проектах.

Примеры представлены в GitLab-репозитории T1 Облако. К каждому примеру кода мы привели комментарии и консольный вывод после выполнения сценария, как делала и автор книги.

Заранее поясним, что в примерах ниже нет секций с настройками провайдера (выбором провайдера, версии) и авторизации. Все эти секции вместе с пояснениями к ним можно найти на главной странице нашего репозитория в разделе «Начало работы» (<https://clck.ru/3FBPy7>).

14.5.1 Пример 1: создание виртуальной машины

Допустим, вы решили создать виртуальную машину. В разделе «Ресурсы и источники данных» нужно выбрать ресурс «t1_compute_instance» (<https://clck.ru/3FBQd4>), описать желаемую инфраструктуру и запустить Terraform следующим образом:

```
data "t1_compute_flavor" "small" { ❶
  vcpus      = 2
  ram        = 4
  family     = "general-purpose"
  cpu_series = "Intel Ice lake 2.8 GHz"
  hardware_group = "public"
}

data "t1_compute_image" "ubuntu" { ❷
  os_distro = "ubuntu"
  os_version = "22.04"
}

data "t1_vpc_network" "default" { ❸
  name = "default"
}

resource "t1_compute_ssh_key" "test" { ❹
  name      = "test-ssh"
  login     = "root"
  public_keys = [
    "ssh-rsa <SSH_KEY_RSA>"
  ]
}

resource "t1_compute_instance" "vm" { ❺
  system_volume = {
    size = 10
  }

  flavor = data.t1_compute_flavor.small ❻
  image  = data.t1_compute_image.ubuntu

  network_interface = {
    subnet_id = data.t1_vpc_network.default.subnets[0].id ❼
  }
  ssh_keys = [
    t1_compute_ssh_key.ssh.id, 8
  ]
}
```

- ❶ Выбираем конфигурацию для ВМ.
- ❷ Описываем, какой образ (дистрибутив, версия) хотим использовать.
- ❸ Выбираем сеть.
- ❹ Создаем SSH-ключи для подключения к ВМ.

- ⑤ Создаем непосредственно VM: `system_volume` – это основной диск, а `size` – его размер в Гб.
- ⑥ Назначаем вышеописанную конфигурацию для VM.
- ⑦ Назначаем подсеть для VM.
- ⑧ Назначаем SSH-ключи для VM.

Выполнение примера

Примените конфигурацию Terraform в терминале. Обязательно введите `yes`, чтобы применить изменения и создать экземпляр: `$ terraform apply`. Вывод консоли будет примерно такой, как ниже.

Вначале Terraform-провайдер читает источники данных:

```
> terraform apply
data.t1_vpc_security_group.sg-default: Reading...
data.t1_vpc_network.default: Reading...
data.t1_compute_flavor.small: Reading...
data.t1_compute_image.ubuntu: Reading...
data.t1_vpc_security_group.sg-default: Read complete after 0s [id=1f8de31c-6ad5-4895-be52-a8f77138d2fe]
data.t1_vpc_network.default: Read complete after 1s [id=94a7a60a-2540-404b-b528-b94a7d62d99b]
data.t1_vpc_subnet.default: Reading...
data.t1_compute_flavor.small: Read complete after 1s [id=a629345a-7a9d-4ec5-936e-7c3f81e8e9bd]
data.t1_vpc_subnet.default: Read complete after 0s [id=a18914fd-c915-496c-b201-ca33cde5e76b]
data.t1_compute_image.ubuntu: Read complete after 1s [id=36939a58-d1b8-498f-a1bb-428211488572]
```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

+ create

Terraform-провайдер показывает, какие действия он выполнит:

Terraform will perform the following actions:

```
# t1_compute_instance.test will be created
+ resource "t1_compute_instance" "test" {
  + allow_delete_volumes = true
  + description           = "Acceptance test instance"
  + flavor                = {
    + cpu_series          = "Intel Cascade Lake 3.0 GHz"
    + family              = "general-purpose"
    + hardware_group      = "public"
    + id                  = "a629345a-7a9d-4ec5-936e-7c3f81e8e9bd"
    + name                = "b3.large.6"
    + ram                 = 12
    + vcpus               = 2
  }
  + id                   = (known after apply)
  + image                = {
```



```

    + id          = "36939a58-d1b8-498f-a1bb-428211488572"
    + name         = "ubuntu_22_04"
    + os_distro    = "ubuntu"
    + os_version   = "22.04"
    + size         = 2361393152
  }
+ name              = (known after apply)
+ network_interface = {
  + fixed_ip        = (known after apply)
  + id              = (known after apply)
  + security_group_ids = [
    + "1f8de31c-6ad5-4895-be52-a8f77138d2fe",
  ]
  + subnet_id       = "a18914fd-c915-496c-b201-ca33cde5e76b"
}
+ region            = "u-central1"
+ ssh_keys          = [
  + (known after apply),
]
+ state             = "off"
+ system_volume     = {
  + disk_type = "Average cluster 3"
  + id       = (known after apply)
  + size     = 4
}
+ zone              = "ru-central1-a"
}

```

```

# t1_compute_ssh_key.test will be created
+ resource "t1_compute_ssh_key" "test" {
  + id          = (known after apply)
  + login       = "root"
  + name        = "test-f123"
  + public_keys = [
    + "ssh-rsa <SSH_KEY_RSA>",
  ]
}

```

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

t1_compute_ssh_key.test: Creating...

t1_compute_ssh_key.test: Creation complete after 1s [id=5ef1b8ec-1511-4f77-8a38-c9834e89c68a]

t1_compute_instance.test: Creating...

t1_compute_instance.test: Still creating... [2m40s elapsed]

t1_compute_instance.test: Creation complete after 2m41s [id=d9c75f1e-5dd1-4019-ba15-d849e6028330]

В результате создано два ресурса: SSH-ключ и ВМ.

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

14.5.2 Пример 2: управление ролями

Мы выбрали этот пример неспроста. Он демонстрирует, что с помощью Terraform можно не только создавать объекты инфраструктуры, но и управлять доступом к ней. Например, мы хотим назначить роль для конкретного сотрудника, то есть выдать ему определенные права и разрешения. Используя Terraform, можно установить как стандартные (администратор, редактор и наблюдатель), так и индивидуальные пользовательские роли. В примере ниже мы выдаем сотруднику пользовательскую роль: он получит разрешение на просмотр списка логов. Смотреть список логов он сможет, а изменять и удалять – нет.

Выбираем ресурс «t1_iam_role» (<https://clck.ru/3FBQm4>) и запускаем такой код:

```
resource "t1_iam_role" "test" { ❶
  name = "test-iam-role-name-6"
  title = "test-iam-role-title-6"
  description = "test-iam-role-description-6"
  organization_name = "test-organization"
  permissions = [
    "auditor:logs:list"
  ]
}

resource "t1_iam_role_attach" "test" { ❷
  user_email = "test-terraform@t1-cloud.ru"
  organization_name = "test-organization"
  role = t1_iam_role.test.name
  role_type = "custom"
}
```

- ❶ Создаем новую роль, даем ей название, описание и список разрешений.
- ❷ Привязываем роль к конкретному пользователю и организации.

Выполнение примера

Этот пример демонстрирует, что чем меньше у ресурса параметров, тем короче будет и сам вывод. В данном случае вывод консоли будет примерно такой:

```
> terraform apply
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
+ create
```

Terraform-провайдер показывает, какие действия он выполнит:

```
Terraform will perform the following actions:
# t1_iam_role.test will be created
+ resource "t1_iam_role" "test" {
  + description      = "test-iam-role-description-6"
```



```

+ name          = "test-iam-role-name-6"
+ organization_name = "test-organization"
+ permissions    = [
    + "auditor:logs:list",
  ]
+ title          = "test-iam-role-title-6"
}
# t1_iam_role_attach.test will be created
+ resource "t1_iam_role_attach" "test" {
  + organization_name = "test-organization"
  + role              = "test-iam-role-name-6"
  + role_type         = "custom"
  + user_email        = "test-terraform@t1-cloud.ru"
}

```

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

t1_iam_role_attach.test: Creating...

t1_iam_role.test: Creating...

t1_iam_role.test: Creation complete after 7s [name=test-iam-role-name-6]

t1_iam_role_attach.test: Creation complete after 8s

В результате создано два ресурса: новая роль и привязка пользователя к этой роли:

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

14.4.3 Пример 3: создание кластера Kubernetes

В разделе «Ресурсы и источники данных» выбираем ресурс «t1_k8s_cluster» (<https://clck.ru/3FBSRL>) и запускаем следующий код:

```

resource "t1_k8s_cluster" "test" { ❶
  description = "Test k8s cluster"
  region      = "ru-central1"
  zone        = "ru-central1-a"
  subnet_id   = data.t1_vpc_subnet.default.id
  k8s_version = "1.28.7"

  control_plane = { ❷
    ha = true
    size = "small"
  }

  security_group_ids = [ ❸
    data.t1_vpc_security_group.sg-default.id
  ]
}

```



```

worker = { ❷
  count = 2
  flavor = data.t1_compute_flavor.small
  pod_cidr = "1.0.0.0/13"
  service_cidr = "1.8.0.0/16"
  add_public_ip = true
  volume = {
    size = 30
    disk_type = "Light"
  }
}

data "t1_vpc_network" "default" { ❸
  name = "default"
}

data "t1_vpc_subnet" "default" { ❹
  network_id = data.t1_vpc_network.default.id
  name = "default-ru-central1"
}

data "t1_vpc_security_group" "sg-default" { ❺
  name = "default"
}

data "t1_compute_flavor" "small" { ❻
  vcpus          = 2
  ram            = 4
  family         = "general-purpose"
  hardware_group = "public"
  cpu_series     = "AMD EPYC 9004 series 4.05 GHz"
}

```

- ❶ Создаем кластер k8s.
- ❷ Настраиваем контрольную панель – компонент k8s.
- ❸ Устанавливаем группы безопасности.
- ❹ Настраиваем рабочие ноды кластера.
- ❺ Выбираем сеть.
- ❻ Выбираем подсеть из выбранной выше сети.
- ❼ Выбираем конкретную группу безопасности.
- ❽ Выбираем флейвор для рабочей ноды.

В результате вывод консоли будет примерно таким.

Вначале Terraform-провайдер читает источники данных:

```

> terraform apply
data.t1_vpc_security_group.sg-default: Reading...
data.t1_vpc_network.default: Reading...
data.t1_compute_flavor.small: Reading...
data.t1_vpc_security_group.sg-default: Read complete after 1s [id=1f8de31c-6ad5-4895-be52-a8f77138d2fe]

```



```
data.t1_vpc_network.default: Read complete after 1s [id=94a7a60a-2540-404b-b528-
b94a7d62d99b]
data.t1_vpc_subnet.default: Reading...
data.t1_compute_flavor.small: Read complete after 1s [id=ca6a5844-a4b7-4833-b731-
a544eb2498f3]
data.t1_vpc_subnet.default: Read complete after 0s [id=a18914fd-c915-496c-b201-
ca33cde5e76b]
```

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:

+ create

Terraform-провайдер показывает, какие действия он выполнит:

Terraform will perform the following actions:

```
# t1_k8s_cluster.test will be created
+ resource "t1_k8s_cluster" "test" {
  + control_plane = {
    + ha = true
    + size = "small"
  }
  + description = "Test k8s cluster"
  + id = (known after apply)
  + k8s_version = "1.28.7"
  + name = (known after apply)
  + region = "ru-central1"
  + security_group_ids = [
    + "1f8de31c-6ad5-4895-be52-a8f77138d2fe",
  ]
  + subnet_id = "a18914fd-c915-496c-b201-ca33cde5e76b"
  + worker = {
    + add_public_ip = true
    + count = 2
    + flavor = {
      + cpu_series = "AMD EPYC 9004 series 4.05 GHz"
      + family = "general-purpose"
      + hardware_group = "public"
      + id = "ca6a5844-a4b7-4833-b731-a544eb2498f3"
      + name = "b4.large.2"
      + ram = 4
      + vcpus = 2
    }
    + pod_cidr = "1.0.0.0/13"
    + service_cidr = "1.8.0.0/16"
    + volume = {
      + disk_type = "Light"
      + size = 30
    }
  }
  + zone = "ru-central1-a"
}
```

Plan: 1 to add, 0 to change, 0 to destroy.


```
Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
t1_k8s_cluster.test: Creating...  
t1_k8s_cluster.test: Still creating... [10s elapsed]  
t1_k8s_cluster.test: Still creating... [24m30s elapsed]
```

В результате создан один ресурс – кластер k8s с двумя нодами:

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

14.6 Заключение

Подход *инфраструктура как код* (infrastructure as code, IaC) преобразил управление инфраструктурой в облачных провайдерах по всему миру. Использование таких инструментов, как Terraform, предназначенный для управления инфраструктурой через декларативные конфигурации, или Crossplane, интегрированный с Kubernetes для управления облачными ресурсами, помогает достичь воспроизводимости результатов, предсказуемости конфигураций и ускорения бизнес-процессов. Это позволяет компаниям быстрее адаптироваться к изменениям на рынке и удовлетворять потребности клиентов.

Полагаем, внедрение IaC становится важным условием для успешного и конкурентоспособного бизнеса в эпоху цифровой трансформации. И вы можете сделать так, чтобы это преимущество работало на вас.

Приложение А

Работа с примерами кода

В этой книге код примеров на языке Python применяется для создания файла конфигурации JSON, который затем использует Terraform. В этом приложении приведены рекомендации по работе с примерами. Почему я выбрала двухэтапный процесс, в ходе которого мы сначала генерируем файл JSON с помощью кода на Python, а затем создаем ресурсы с помощью Terraform?

Во-первых, я хотела сделать так, чтобы каждый, кто хочет запустить примеры, но не может использовать Google Cloud Platform (GCP), имел возможность изучить их дальше. Это позволяет осуществлять «локальную» разработку и тестирование, а также, при необходимости, создавать реально действующие ресурсы инфраструктуры.

Во-вторых, файлы JSON очень объемные! В виде листингов они выглядят многословными и запутанными. Оболочка в виде кода Python позволяет мне демонстрировать примеры паттернов, не пробираясь через строки конфигурации JSON. Добавление кода Python к синтаксису Terraform JSON дает некоторую уверенность в будущем на случай, если мне придется переписать примеры в другом инструменте.

ПРИМЕЧАНИЕ Справочные ссылки, библиотеки и синтаксис инструментов постоянно меняются. Самую последнюю версию кода можно скачать по адресу <https://github.com/joatmon08/manning-book>.

На рис. А.1 показан рабочий процесс, необходимый для запуска примеров. После выполнения команды `python main.py` вы получи-

те файл JSON с расширением `.tf.json`. Выполните команду `terraform init` в интерфейсе командной строки, чтобы инициализировать инструментальное состояние, и `terraform apply` для конфигурирования ресурсов.

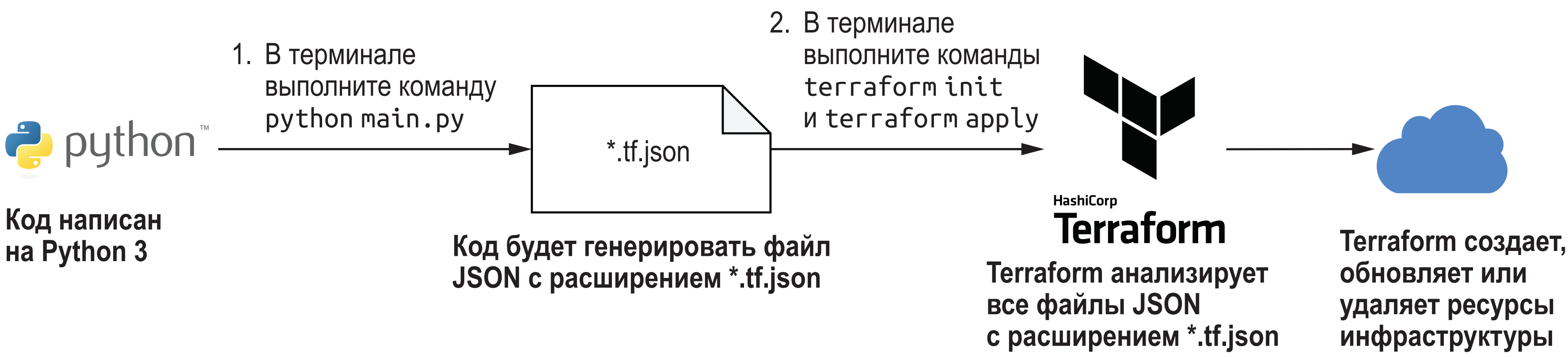


Рис. А.1 Используйте код на Python для создания файла JSON и примените конфигурацию с помощью Terraform

Я кратко расскажу, как настроить учетную запись у различных облачных провайдеров. Затем представлю Python и библиотеки, на которые ссылаюсь в примерах, такие как доступ к API инфраструктуры и тестирование. Наконец, я кратко объясню, как использовать Terraform с GCP.

А.1 Провайдеры облачных услуг

Примеры этой книги ориентированы на услуги облачного провайдера Google Cloud Platform (GCP). Если вы предпочитаете другого провайдера, во многих примерах есть пояснения об эквивалентной реализации для получения аналогичной архитектуры. В табл. А.1 показано приблизительное соответствие между разными типами ресурсов в GCP, Amazon Web Services (AWS) и Microsoft Azure.

Таблица А.1 Сопоставление ресурсов между провайдерами облачных услуг

Ресурс	GCP	AWS	Azure
Группировка ресурсов	Проект Google	Учетная запись AWS	Подписка Azure и группы ресурсов
Идентификация и управление доступом (IAM)	Google IAM	AWS IAM	Azure Active Directory
Серверы Linux (Ubuntu)	Вычислительный экземпляр Google	Экземпляр Amazon EC2	Виртуальная машина Azure Linux
Организация сети	Google Virtual Private Cloud (VPC) Подсети Есть сеть по умолчанию	Amazon Virtual Private Cloud (VPC) Подсети Таблицы маршрутизации Шлюзы Есть сеть по умолчанию	Azure Virtual Network Подсети Ассоциированные таблицы маршрутизации
Правила брандмауэра	Firewall rules	Группы безопасности Списки прав сетевого доступа	Сетевые группы безопасности

Таблица А.1 (окончание)

Ресурс	GCP	AWS	Azure
Балансировка нагрузки	Правила перенаправления нагрузки Google (L4) Балансировщик нагрузки HTTP(S) (L7)	Балансировщик AWS Elastic Load Balancing (ELB) (L4) AWS Application Load Balancer (ALB) (L7)	Azure Load Balancer (L4) Azure Application Gateway (L7)
Реляционная БД	Google Cloud SQL	Amazon Relational Database Service (RDS)	Azure Database PostgreSQL
Оркестратор контейнеров (Kubernetes)	Google Kubernetes Engine (GKE)	Amazon Elastic Kubernetes Service (EKS)	Azure Kubernetes Service (AKS)

В этом разделе я опишу некоторые первоначальные настройки, которые вам нужно будет выполнить для выбранного вами провайдера облачных услуг.

А.1.1 Облачная платформа Google

Перед началом использования GCP создайте новый проект (<http://mng.bz/mOV2>) и запускайте все примеры в этом проекте. Это позволит вам удалить проект и его ресурсы, когда вы закончите читать книгу.

Затем установите интерфейс командной строки (CLI) `gcloud` (<https://cloud.google.com/sdk/docs/install>). CLI поможет вам пройти аутентификацию, чтобы Terraform получил доступ к GCP API:

```
$ gcloud auth application-default login
```

Эта команда настраивает учетные данные на вашем компьютере, чтобы Terraform мог аутентифицироваться в GCP (<http://mng.bz/5Qw1>).

А.1.2 Amazon Web Services

Перед началом использования AWS создайте новую учетную запись (<http://mng.bz/6XDD>) и запускайте все примеры в этой учетной записи. Это позволит вам удалить учетную запись и ее ресурсы, когда вы закончите читать книгу.

Далее создайте набор ключей доступа в консоли AWS (<http://mng.bz/o21r>). Вам нужно будет сохранить эти ключи, чтобы Terraform в дальнейшем смог получить доступ к API AWS.

Скопируйте идентификатор ключа доступа и секретный ключ доступа и сохраните их в переменных среды:

```
$ export AWS_ACCESS_KEY_ID=<Access key ID>
$ export AWS_SECRET_ACCESS_KEY=<Secret access key>
```

Затем укажите регион AWS, который вы хотите использовать:

```
$ export AWS_REGION="us-east-1"
```


Эта последовательность команд настраивает учетные данные на вашем компьютере, чтобы Terraform мог пройти аутентификацию в AWS (<http://mng.bz/nNWg>).

A.1.3 *Microsoft Azure*

Перед началом использования Azure создайте новую учетную запись (<http://mng.bz/v6nJ>). Создание новой учетной записи по умолчанию дает вам подписку. Вы сможете создавать ресурсы внутри подписки и распределять их по группам. После прочтения книги вы можете удалить группу ресурсов.

Затем установите Azure CLI (<http://mng.bz/44Da>). CLI поможет вам пройти аутентификацию, чтобы Terraform получил доступ к API Azure.

Войдите в Azure CLI:

```
$ az login
```

Запросите перечень подписок, чтобы получить идентификатор подписки по умолчанию:

```
$ az account list
```

Скопируйте идентификатор подписки и сохраните его в переменной среды:

```
$ export ARM_SUBSCRIPTION_ID="<subscription ID>"
```

Эта последовательность команд настраивает учетные данные на вашем компьютере, чтобы Terraform мог пройти аутентификацию в Azure (<http://mng.bz/QvPw>). Для каждого примера необходимо создать группу ресурсов Azure (<http://mng.bz/XZNG>). Чтобы удалить все ресурсы инфраструктуры, созданные для использования в примере, необходимо удалить соответствующую группу.

A.2 *Python*

Прежде чем приступить к выполнению примеров, необходимо скачать Python. Листинги кода предназначены для версии Python 3. Вы можете установить Python несколькими способами, например используя выбранный вами менеджер пакетов или страницу загрузок Python (www.python.org/downloads/). Однако я предпочитаю использовать `pyenv` (<https://github.com/pyenv/pyenv>) для загрузки и управления своими версиями Python. Менеджер `pyenv` позволяет вам выбрать нужную вам версию Python и установить ее в виртуальную среду с помощью библиотеки Python `venv` (<https://docs.python.org/3/library/venv.html>).

Я использую виртуальную среду, потому что у меня много проектов, требующих разных версий Python. Установка разных версий каждого проекта в одной и той же среде сбивает с толку и часто приводит к порче кода. Поэтому я выделяю каждый проект в отдельную среду разработки со своими зависимостями и версией Python.

A.2.1 Установка библиотек Python

После установки Python 3 в вашу среду разработки или виртуальную среду вам необходимо установить некоторые внешние библиотеки. В листинге A.1 я перечисляю библиотеки и зависимости в текстовом файле requirements.txt со списком пакетов и версий.

Листинг A.1 Файл requirements.txt с библиотеками Python для этой книги

```
apache-libcloud==3.3.1           ①
google-api-python-client==2.17.0 ②
google-cloud-billing==1.3.3       ②
netaddr==0.8.0                   ③
pytest==6.2.4                     ④
```

- ① Устанавливаем библиотеку Apache Libcloud.
- ② Устанавливаем клиентские библиотеки для GCP, включая клиента Python и клиента Cloud Billing.
- ③ Устанавливаем библиотеку Python netaddr для анализа сетевой информации.
- ④ Устанавливаем фреймворк тестирования Python pytest.

Репозиторий примера содержит файл requirements.txt, который задает необходимые версии библиотек для установки. В нужной среде разработки Python воспользуйтесь CLI для установки библиотек с помощью установщика пакетов Python pip:

```
$ pip install -r requirements.txt
```

Некоторые примеры требуют более сложной автоматизации или тестирования. Они ссылаются на библиотеки, которые вам нужно будет импортировать отдельно. Давайте рассмотрим эти библиотеки более подробно.

Apache Libcloud

Apache Libcloud (<https://libcloud.apache.org/>) реализует интерфейс Python для создания, обновления, чтения и удаления облачных ресурсов. Он предлагает единый интерфейс, независимый от облачной службы или провайдера. Я импортирую эту библиотеку в первых разделах книги в примерах по интеграции и сквозному тестированию. Чтобы использовать Apache Libcloud, импортируйте пакет libcloud, как показано в листинге A.2, и настройте драйвер для подключения к GCP.

Листинг А.2 Импорт Apache Libcloud

```

from libcloud.compute.types import Provider ❶
from libcloud.compute.providers import get_driver ❷

ComputeEngine = get_driver(Provider.GCE) ❸
driver = ComputeEngine( ❹
    credentials.GOOGLE_SERVICE_ACCOUNT, ❹
    credentials.GOOGLE_SERVICE_ACCOUNT_FILE, ❹
    project=credentials.GOOGLE_PROJECT, ❹
    datacenter=credentials.GOOGLE_REGION) ❹

```

- ❶ Импортируем объект для настройки провайдера облачных услуг, например GCP.
- ❷ Импортируем функцию инициализации драйвера для провайдера облачных услуг.
- ❸ Настраиваем драйвер, который будет подключаться к Google Cloud.
- ❹ Передаем учетные данные для подключения к Google Cloud API для инициализации драйвера.

В тестах я использую Apache Libcloud вместо клиентских библиотек Google Cloud, поскольку он предоставляет унифицированный API для доступа к любому облаку. Если я захочу переключить свои примеры на AWS или Azure, мне придется изменить лишь драйвер облачного провайдера. Тесты только считывают информацию от провайдера и не выполняют каких-либо сложных операций с Apache Libcloud.

Эквиваленты AWS и Azure

Вам потребуется обновить драйвер Apache Libcloud, чтобы использовать драйвер Amazon EC2 (<http://mng.bz/yvQG>) или Azure ARM Compute Driver (<http://mng.bz/M5B7>).

Клиенты Python для Google Cloud

В последней части книги представлены более сложные IaC и тесты, которые мне не удалось реализовать в Apache Libcloud. Этот фреймворк не поддерживает мой способ получения конкретной информации о ресурсах Google Cloud, например информации о ценах. В листинге А.3 показано, как в этих случаях я использовала клиентские библиотеки, специфичные для Google Cloud.

Листинг А.3 Импорт клиентской библиотеки Google Cloud

```

import googleapiclient.discovery ❶
from google.cloud import billing_v1 ❷

```

- ❶ Импортируем клиентскую библиотеку Google Cloud для Python.
- ❷ Импортируем клиента Python для Google Cloud Billing API.

Эквиваленты AWS и Azure

Импортируйте AWS SDK для Python (<https://aws.amazon.com/sdk-for-python/>) или библиотеку Azure для Python (<http://mng.bz/VMV0>), чтобы создавать примеры в AWS или Azure.

В примерах используются две библиотеки, поддерживаемые Google Cloud. Клиентская библиотека Google Cloud для Python (<http://mng.bz/aJ1z>) позволяет получить доступ ко многим API Google Cloud, а также создавать, читать, обновлять и удалять ресурсы. Однако она не поддерживает доступ к API облачного биллинга Google Cloud Billing.

В результате для главы 12 о стоимости мне пришлось импортировать другую библиотеку, поддерживаемую Google Cloud, для получения информации из каталога биллинга. Клиент Python для Google Cloud Billing API (<http://mng.bz/gwBl>) позволяет мне читать информацию из каталога сервисов Google Cloud.

Если вашей IaC необходимо обращаться к определенным ресурсам или API, недоступным в унифицированном API, например Apache Libcloud, то, скорее всего, вам придется искать отдельную библиотеку для получения необходимой информации. Хотя нам хотелось бы свести к минимуму зависимости, приходится признать, что не каждая библиотека подходит для всех вариантов использования! Выберите другую библиотеку, если вы видите, что существующая не может обеспечить необходимую вам автоматизацию.

Netaddr

В главе 5 мне нужно было изменить блок IP-адресов. Размышляя о возможности математического расчета правильного адреса, я решила воспользоваться готовой библиотекой. В Python есть встроенная библиотека `ipaddress`, но она не поддерживает необходимую мне функциональность. Вместо нее я установила `netaddr` (<https://netaddr.readthedocs.io/en/latest/>), чтобы уменьшить объем дополнительного кода, необходимого для расчета IP-адресов.

Pytest

Многие тесты в этой книге используют `pytest` – фреймворк тестирования Python. Вы также можете воспользоваться модулем Python `unittest` для разработки и запуска тестов. Я предпочитаю `pytest`, поскольку он снабжен минимальным интерфейсом для написания и запуска тестов без более сложных функций тестирования. Я не буду подробно описывать `pytest` и ограничусь упоминанием функций, которые использую в тестах, и способами их запуска.

`Pytest` ищет файлы Python с префиксом `test_`. Это имя файла сигнализирует о том, что файл содержит тесты Python. Каждая тестовая

функция также использует префикс `test_`. Pytest выбирает и запускает тесты на основе префикса.

Многие из тестов в этой книге используют функциональность *тестового стенда* (test fixture). Тестовый стенд отслеживает определенный объект, например имя или константу, который можно использовать для сравнения в нескольких тестах. В листинге А.4 я использую тестовый стенд для передачи часто обрабатываемых объектов, таких как сетевые атрибуты, между несколькими тестами.

Листинг А.4 Пример использования pytest

```
import pytest 1

@pytest.fixture 2
def network(): 3
    return 'my-network' 3

def test_configuration_for_network_name(network): 3
    assert network == 'my-network', 'Имя сети не соответствует ожидаемому' 4
```

- ❶ Импортируем библиотеку pytest.
- ❷ Задаем известный объект или тестовый стенд.
- ❸ Возвращаем известное имя сети 'my-network' и передаем его первому тесту.
- ❹ Убеждаемся, что имя сети соответствует ожидаемому, в противном случае тест не пройден. Можно добавить информативное сообщение об ошибке.

Самая важная часть теста заключается в проверке совпадения ожидаемого значения с фактическим (asserting). Pytest предлагает одного оператора `assert` для каждого теста. Я следую этому соглашению, потому что оно помогает мне писать более описательные и полезные тесты. Ваши тесты должны максимально четко описывать ваши намерения и то, что они проверяют.

Чтобы запустить набор тестов с помощью pytest, вы можете передать каталог с тестами. Но сначала убедитесь, что в вашем каталоге тестов указаны абсолютные пути ко всем файлам, которые вы читаете с помощью pytest! Например, тесты в главе 4 читают внешние файлы JSON. Поэтому вам необходимо перейти в рабочий каталог с номером главы и раздела:

```
$ cd ch05/s02
```

Вы можете запустить все тесты в каталоге, передав точку (.) в pytest в командной строке:

```
$ pytest .
```

Для запуска одного файла передайте его имя в pytest в командной строке:

```
$ pytest test_network.py
```


Во многих тестах в этой книге используются схожие паттерны тестовых стендов и проверок. Для получения дополнительной информации о других функциях `pytest` ознакомьтесь с его документацией (<https://docs.pytest.org>). Для запуска примеров выполните команды `pytest` или `python main.py` в своем терминале командной строки.

A.2.2 *Запуск Python*

Я разделяю каждый ресурс инфраструктуры по файлам Python. Каждый каталог содержит файл `main.py`, как показано в листинге A.5. Файл всегда содержит код, который записывает словарь Python в файл JSON. Объект должен использовать синтаксис конфигурации JSON Terraform для описания ресурса инфраструктуры.

Листинг A.5 Пример файла `main.py`, записывающего словарь в файл JSON

```
import json
if __name__ == "__main__":
    server = ServerFactoryModule(name='hello-world')
    with open('main.tf.json', 'w') as outfile:
        json.dump(server.resources, outfile, sort_keys=True, indent=4)
```

- ❶ Генерируем словарь Python для сервера GCP.
- ❷ Создаем файл JSON с именем `'main.tf.json'`, который содержит конфигурацию JSON, совместимую с Terraform.
- ❸ Записываем словарь сервера в файл JSON.

Для запуска скрипта Python введите в своем терминале команду

```
$ python main.py
```

Просмотрев содержимое текущего каталога, вы найдете новый файл JSON с именем `main.tf.json`:

```
$ ls
main.py main.tf.json
```

Во многих примерах требуется запустить Python для выполнения `main.py` и создать файл JSON с именем `main.tf.json`, если не указано иное. Однако в некоторых примерах для автоматизации либо тестирования применяются другие библиотеки или код.

A.3 *HashiCorp Terraform*

После создания файла `main.tf.json` с помощью `python main.py` вам необходимо создать ресурсы в GCP. Файлам `.tf.json` требуется HashiCorp Terraform для создания, чтения, обновления и удаления ресурсов в GCP.

Загрузите и установите Terraform с помощью любого менеджера пакетов (www.terraform.io/downloads.html). Для запуска Terraform необходимо ввести команду в терминале, поэтому вам нужно будет загрузить бинарный файл и убедиться, что вы можете запустить его в своем терминале. Terraform ищет файлы с расширением .tf или .tf.json в рабочем каталоге и создает, читает, обновляет и удаляет ресурсы, которые вы определяете в этих файлах.

A.3.1 Синтаксис конфигурации JSON

Terraform предлагает различные интерфейсы для создания ресурсов инфраструктуры. В большей части документации используется язык конфигурации HashiCorp Configuration Language (HCL) – это предметно-ориентированный язык (DSL), который определяет ресурсы инфраструктуры для каждого провайдера облачных услуг. Дополнительную информацию о Terraform можно найти в документации (www.terraform.io/docs/index.html).

В примерах этой книги я не использую HCL. Примеры основаны на синтаксисе конфигурации JSON (www.terraform.io/docs/language/syntax/json.html), специфичном для Terraform. Этот синтаксис представляет собой тот же предметно-ориентированный язык, что и HCL, только отформатированный в JSON.

Каждый файл main.py в Python записывает словарь в файл JSON. В листинге A.6 показано создание словаря, определяющего ресурс Terraform в синтаксисе конфигурации JSON. Ресурс JSON обращается к ресурсу google_compute_instance, определенному Terraform (<http://mng.bz/e71z>), и устанавливает все необходимые атрибуты.

Листинг A.6 Словарь Python для сервера в Terraform JSON

```
terraform_json = {
    'resource': [{
        'google_compute_instance': [{
            'my_server': [{
                'allow_stopping_for_update': True
                'boot_disk': [{
                    'initialize_params': [{
                        'image': 'ubuntu-1804-lts'
                    }]
                }],
                'machine_type': 'e2-micro',
                'name': 'my-server',
                'zone': 'us-central1-a',
            ]
        }],
    }],
}
```

❶ Сообщаем Terraform, что это определение списка ресурсов.

- ② Определяем ресурс Terraform 'google_compute_instance', который создаст и настроит сервер в GCP.
- ③ Определяем уникальный идентификатор сервера, чтобы Terraform мог его отслеживать.

Эквиваленты AWS и Azure

В AWS вы должны использовать ресурс Terraform `aws_instance` со ссылкой на VPC по умолчанию (<http://mng.bz/pOPG>).

В Azure вам потребуется создать виртуальную сеть и подсети. Затем создайте ресурс Terraform `azurerm_linux_virtual_machine` (<http://mng.bz/Ooxn>) в сети.

Словарь Python становится описанием конфигурации Terraform JSON, когда вы записываете его в файл JSON с соответствующим синтаксисом. Terraform создаст ресурсы, определенные в своем текущем рабочем каталоге, принимая во внимание только файлы с расширением `.tf` или `.tf.json`. Если вы сохраните код конфигурации в файл JSON, который не имеет расширения `.tf.json`, Terraform не распознает ресурсы в файле.

A.3.2 Инициализация состояния

После запуска Python и создания файла JSON вам необходимо инициализировать Terraform в рабочем каталоге. На рис. A.2 показаны команды, которые необходимо выполнить в терминале для инициализации состояния и применения изменений инфраструктуры.

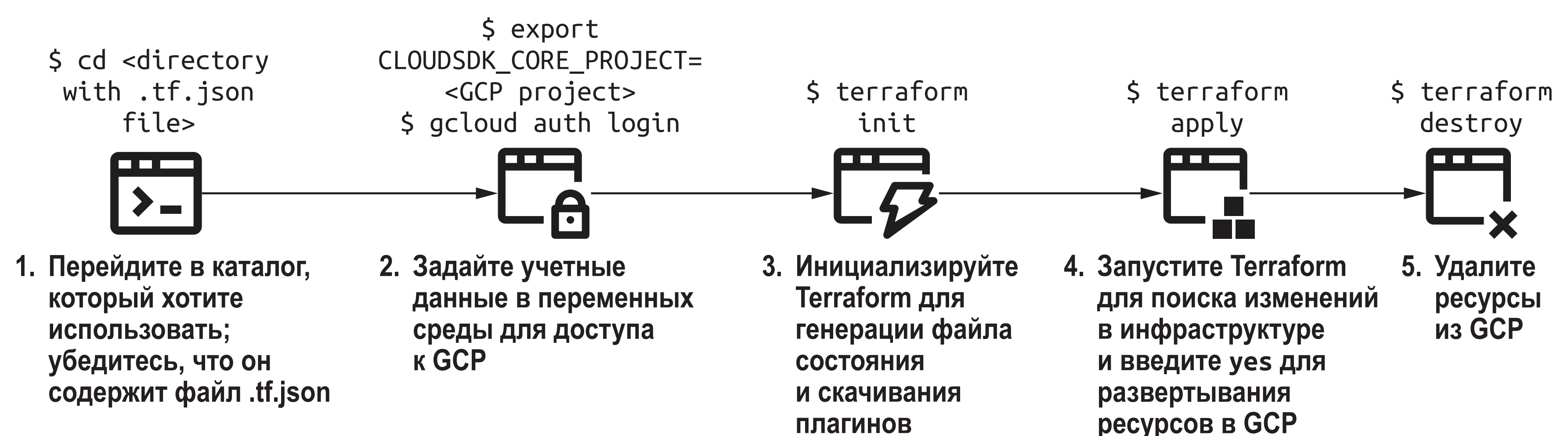


Рис. A.2 Инициализация и развертывание ресурсов с помощью Terraform в рабочем каталоге. Удалите ресурсы, когда закончите работу с примером

В терминале перейдите в каталог с файлом `*.tf.json`. Например, я перехожу в каталог, содержащий примеры для раздела 2.3:

```
$ cd ch02/s03
```


Инициализируйте Terraform в своем терминале:

```
$ terraform init
Initializing the backend...
```

```
Initializing provider plugins...
```

- Reusing previous version of hashicorp/google from the dependency lock file
- Using previously-installed hashicorp/google v3.86.0

```
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running "terraform plan"
```

```
➔to see any changes that are required for your infrastructure.
```

```
➔All Terraform commands should now work.
```

```
If you ever set or change modules or backend configuration
```

```
➔for Terraform, rerun this command to reinitialize
```

```
➔your working directory. If you forget, other
```

```
➔commands will detect it and remind you to do so if necessary.
```

Terraform выполняет этап инициализации, который создает инструментальное состояние, называемое *бэкендом* (backend), и устанавливает плагины и модули. При инициализации создается ряд файлов, которые не следует удалять из файловой системы. После инициализации Terraform вы обнаружите несколько скрытых и новых файлов, просмотрев содержимое текущего каталога:

```
$ ls -al
drwxr-xr-x .terraform
-rw-r--r-- .terraform.lock.hcl
-rw-r--r-- main.py
-rw-r--r-- main.tf.json
-rw-r--r-- terraform.tfstate
-rw-r--r-- terraform.tfstate.backup
```

Terraform сохраняет инструментальное состояние в специальном файле состояния, чтобы быстро отражать любые изменения, вносимые вами в ресурсы инфраструктуры. Terraform может обращаться к файлу состояния, хранящемуся локально или на сервере, в реестре артефактов, в хранилище объектов и т. д. В примерах состояние инструмента сохраняется в локальном файле с именем terraform.tfstate. Если вы случайно удалите этот файл, Terraform больше не сможет распознавать ресурсы, находящиеся под его управлением! Не удаляйте локальный файл состояния и не редактируйте примеры для использования удаленного бэкенда. В каталоге вы также найдете файл terraform.tfstate.backup, который Terraform использует для резервного копирования инструментального состояния перед внесением изменений.

При инициализации также устанавливается плагин Terraform для связи с Google. Terraform использует систему плагинов для расширения функциональности своего движка и взаимодействия с провайдерами облачных услуг. В примерах AWS используется одна и та же команда `terraform init` для автоматической загрузки плагина AWS. Плагины или модули загружаются в каталог `.terraform`.

Terraform также фиксирует версии плагинов, аналогично файлу `require.txt` для Python. Список фиксированных версий плагинов вы найдете в `.terraform.lock.hcl`. В репозитории примеров я разместила `.terraform.lock.hcl` в системе контроля версий, чтобы Terraform устанавливал только те плагины, с которыми я работала на момент создания примеров.

A.3.3 *Настройка учетных данных в терминале*

Большинство плагинов Terraform получают учетные данные для API провайдера инфраструктуры из переменных среды. Обычно я задаю переменную среды проекта GCP, поэтому Terraform подключается к правильному проекту GCP:

```
$ export CLOUDSDK_CORE_PROJECT=<your GCP project ID>
```

Я также прохожу аутентификацию в GCP с помощью CLI инструмента `gcloud`. Следующая команда автоматически устанавливает учетные данные Terraform для доступа к GCP:

```
$ gcloud auth login
```

При работе с другими провайдерами облачных услуг я рекомендую настроить переменные среды в вашем терминале для аутентификации в вашей учетной записи AWS или Azure. О получении учетных записей говорилось в разделе A.1.

A.3.4 *Развертывание ресурсов с помощью Terraform*

После настройки учетных данных вы можете использовать Terraform для пробного прогона и развертывания ресурсов вашей инфраструктуры. Выполните в окне терминала команду `terraform apply`, чтобы начать развертывание ваших изменений:

```
$ terraform apply
```

```
Terraform used the selected providers to generate
➡the following execution plan. Resource actions
➡are indicated with the following symbols:
+ create
```


Terraform will perform the following actions:

```
# google_compute_instance.hello-world will be created
+ resource "google_compute_instance" "hello-world" {
```

```
... СОКРАЩЕНО ...
```

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value:

Команда остановится и будет ждать, пока вы введете `yes` (да) после запроса `Enter a value:` (введите значение). Terraform ждет, пока вы просмотрите изменения и подтвердите, что вы хотите добавить, изменить или удалить ресурсы. Всегда тщательно проверяйте изменения, прежде чем вводить `yes`!

После того как вы введете `yes`, Terraform начнет развертывание ресурсов:

Enter a value: yes

```
google_compute_instance.hello-world: Creating...
```

```
google_compute_instance.hello-world:
```

```
➔Still creating... [10s elapsed]
```

```
google_compute_instance.hello-world:
```

```
➔Creation complete after 15s [id=projects/infrastructure-as-code-book/zones
```

```
➔/us-central1-a/instances/hello-world]
```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

После выполнения команды `terraform apply` вы найдете новые ресурсы в своем проекте GCP.

A.3.5 Удаление ресурсов

Во многих примерах из этой книги используются совпадающие имена или сетевые блоки CIDR. Я рекомендую вам удалять ресурсы при переходе к следующей главе или разделу. Terraform использует команду `terraform destroy`, чтобы удалить все ресурсы, перечисленные в `terraform.tfstate`, из GCP. В своем терминале убедитесь, что вы прошли аутентификацию в GCP или поставщике инфраструктуры.

Если вы запустите команду `terraform destroy`, она выведет список ресурсов, подлежащих удалению. Проверьте список и убедитесь, что вы действительно хотите удалить эти ресурсы!

```
$ terraform destroy
```


Terraform used the selected providers to generate
➡the following execution plan. Resource actions
➡are indicated with the following symbols:
- destroy

Terraform will perform the following actions:

```
# google_compute_instance.hello-world will be destroyed
```

```
... СОКРАЩЕНО ...
```

```
Plan: 0 to add, 0 to change, 1 to destroy.
```

Do you really want to destroy all resources?

Terraform will destroy all your managed infrastructure,
➡as shown above.

There is no undo. Only 'yes' will be accepted to confirm.

Enter a value:

После тщательной проверки введите `yes` в командной строке. Terraform удалит ресурсы из GCP. Удаление займет некоторое время, поэтому будьте готовы ждать несколько минут. Некоторые примеры займут еще больше времени для развертывания и уничтожения, поскольку для них требуется много ресурсов:

Enter a value: yes

```
google_compute_instance.hello-world: Destroying... elapsed]
google_compute_instance.hello-world: Still destroying...
➡[id=projects/infrastructure-as-code-book/zones
➡/us-central1-a/instances/hello-world, 2m10s elapsed]
google_compute_instance.hello-world: Destruction complete after 2m24s
```

```
Destroy complete! Resources: 1 destroyed.
```

После удаления ресурсов вы можете удалить файлы `terraform.tfstate`, `terraform.tfstate.backup` и `.terraform`, если хотите. Не забывайте удалять свои ресурсы из GCP (или удалять весь проект) каждый раз, когда завершаете работу с примером, чтобы сократить расходы на облако!

Приложение В

Решения к упражнениям

Упражнение 1.1

Выберите скрипт или конфигурацию инфраструктуры в своей организации. Оцените их соответствие принципам IaC. Способствуют ли они воспроизводимости и идемпотентности, помогают ли в компоновке и облегчают ли развитие?

Ответ

Выполните следующие действия, чтобы определить, соответствует ли скрипт или конфигурация вашей инфраструктуры принципам IaC:

- *воспроизводимость* – скопируйте и вставьте сценарий или конфигурацию, поделитесь ими с кем-нибудь еще и попросите их создать ресурсы, не изменяя конфигурацию;
- *идемпотентность* – запустите сценарий несколько раз. Это не должно изменить вашу инфраструктуру;
- *комбинируемость* – скопируйте часть конфигурации и создайте ее поверх других ресурсов инфраструктуры;
- *развиваемость* – добавьте в конфигурацию новый ресурс инфраструктуры и убедитесь, что вы можете вносить изменения, не затрагивая другие ресурсы.

Упражнение 2.1

Какой стиль настройки инфраструктуры используется в следующем примере – императивный или декларативный?


```
if __name__ == "__main__":
    update_packages()
    read_ssh_keys()
    update_users()
    if enable_secure_configuration:
        update_ip_tables()
```

Ответ

Фрагмент кода использует императивный стиль настройки инфраструктуры. Он определяет, как настроить сервер шаг за шагом в определенной последовательности, а не объявляет конкретную целевую конфигурацию.

Упражнение 2.2

Какие из следующих изменений выигрывают от принципа неизменности (выберите все подходящие варианты):

- a** сокращение сети для уменьшения количества IP-адресов;
- b** добавление столбца в реляционную базу данных;
- c** добавление нового IP-адреса в существующую запись DNS;
- d** обновление пакетов сервера до обратно несовместимых версий;
- e** миграция ресурсов инфраструктуры в другой регион.

Ответ

Правильные варианты – a, d и e. Каждое изменение выигрывает от нового набора ресурсов, реализующих это изменение. Если вы попытаетесь внести изменения прямо в работающие ресурсы, то можете случайно вывести из строя промышленную систему. Например, сокращение сети с целью уменьшения количества IP-адресов может привести к выпадению за пределы диапазона некоторых ресурсов, использующих сеть.

Обновление пакетов до обратно несовместимых версий означает, что неработающее обновление пакета может повлиять на способность сервера обрабатывать запросы пользователей. Миграция инфраструктурных ресурсов в другой регион требует времени. Не все регионы провайдеров облачных услуг поддерживают полный перечень типов ресурсов. Создание новых ресурсов помогает смягчить проблемы с миграцией. Вы можете вносить изменения, указанные в вариантах b и c, попеременно без особого риска повлиять на систему.

Упражнение 3.1

К каким паттернам модулей применим следующий IaC (выберите все подходящие варианты):

```
if __name__ == "__main__":
    environment = 'development'
    name = f'{environment}-hello-world'
```



```
cidr_block = '10.0.0.0/16'

# NetworkModule возвращает подсеть или сеть
network = NetworkModule(name, cidr_block)

# TagsModule возвращает список тегов по умолчанию
tags = TagsModule0

# ServerModule возвращает одиночный сервер
server = ServerModule(name, network, tags)
```

- a** фабричный паттерн;
- b** паттерн «одиночка»;
- c** паттерн «прототип»;
- d** паттерн «строитель»;
- e** паттерн «компоновщик».

Ответ

К данной IaC применимы варианты а, с и е. Сетевой модуль использует паттерн «компоновщик» для создания подсети и сети. Модуль тегов использует паттерн «прототип» для возврата метаданных состояния. Модуль сервера использует фабричный паттерн для возврата одного сервера на основе имени, сети и тегов.

В коде не используются паттерны «одиночка» или «строитель». Он не создает единые глобальные ресурсы или внутреннюю логику для создания конкретных ресурсов.

Упражнение 4.1

Каким наилучшим способом можно смягчить зависимость базы данных от сети с помощью следующего кода IaC:

```
class Database:
    def __init__(self, name):
        spec = {
            'name': name,
            'settings': {
                'ip_configuration': {
                    'private_network': 'default'
                }
            }
        }
```

- a** этот подход достаточно хорошо смягчает зависимость базы данных от сети;
- b** нужно передать идентификатор сети как переменную, а не жестко программировать его по умолчанию;
- c** реализуйте и передайте объект NetworkOutput в модуль базы данных для всех сетевых атрибутов;

- d** добавьте в сетевой модуль функцию для передачи его сетевого идентификатора в модуль базы данных;
- e** добавьте в модуль базы данных функцию для получения через API инфраструктуры идентификатора сети по умолчанию.

Ответ

Вы можете реализовать паттерн «адаптер» для вывода сетевых атрибутов (с). База данных выбирает, какие сетевые атрибуты она использует, например идентификатор сети или блок CIDR. Этот подход лучше всего соответствует принципу внедрения зависимостей. Хотя вариант d реализует инверсию зависимостей, он не реализует инверсию управления. Вариант e реализует внедрение зависимостей, но по-прежнему содержит жестко запрограммированный идентификатор сети.

Упражнение 6.1

Допустим, вы заметили, что новая версия модуля балансировки нагрузки нарушает вашу конфигурацию DNS. Коллега по команде обновил модуль, чтобы он выдавал частные IP-адреса вместо общедоступных. Что вы можете сделать, чтобы команда твердо усвоила, что модулю нужны общедоступные IP-адреса:

- a** создать отдельный модуль балансировки нагрузки для частных IP-адресов;
- b** добавить контрактные тесты модуля, чтобы убедиться, что модуль выводит как частные, так и общедоступные IP-адреса;
- c** обновить документацию модуля, указав, что ему необходимы общедоступные IP-адреса;
- d** запустить интеграционные тесты модуля и убедиться, что IP-адрес общедоступен.

Ответ

Вместо создания нового модуля вы можете добавить контрактный тест, чтобы помочь команде запомнить, что ресурсам высокого уровня нужны как частные, так и общедоступные IP-адреса (b). Вы можете создать отдельный модуль балансировки нагрузки (a). Однако это не обязательно поможет вашей команде запомнить, что определенные модули должны выводить определенные переменные. Обновление документации модуля (c) означает, что ваша команда должна не забыть сначала прочитать документацию. Интеграционные тесты требуют временных и финансовых затрат на выполнение (d), в то время как контрактные тесты достаточно хорошо устраняют проблему.

Упражнение 6.2

Допустим, вы добавляете правила брандмауэра, чтобы разрешить приложению доступ к новой очереди. Какая комбинация тестов с точки зрения вашей команды будет наиболее полезной для внесения изменений:

- a модульные и интеграционные тесты;
- b контрактные и сквозные тесты;
- c контрактные и интеграционные тесты;
- d модульные и сквозные тесты.

Ответ

Двумя наиболее ценными тестами будут модульные и сквозные тесты (d). Модульные тесты дают уверенность, что кто-то не отменит новые правила в будущем. Сквозные тесты проверяют, может ли приложение успешно получить доступ к очереди. Контрактные тесты не принесут никакой пользы, поскольку вам не нужно проверять входные и выходные данные для правил брандмауэра.

Упражнение 7.1

Выберите стандартное изменение инфраструктуры в вашей организации. Что вам нужно для того, чтобы уверенно и непрерывно внедрять изменения в производство? Как насчет непрерывного развертывания? Схематически изобразите этапы ваших конвейеров поставки в обоих случаях.

Ответ

Выполняя это упражнение, обратите внимание на следующие моменты:

- есть ли у вас модульные, интеграционные или сквозные тесты?
- какую модель ветвления вы используете?
- есть ли в вашей компании политика комплаенса? Например, два человека должны утвердить обновление перед развертыванием в производстве;
- что происходит, когда кому-то нужно внести изменения?

Упражнение 9.1

Рассмотрим следующий код:

```
if __name__ == "__main__":
    network.build()
    queue.build(network)
    server.build(network, queue)
    load_balancer.build(server)
    dns.build(load_balancer)
```


Очередь зависит от сети. Сервер зависит от сети и очереди. Как бы вы выполнили blue-green развертывание для обновления очереди с помощью SSL?

Ответ

Вы создаете очередь green с включенным SSL. Затем создаете новый сервер green, который зависит от очереди. Проверьте, что приложение на этом сервере может получить доступ к очереди с конфигурацией SSL. Если тест пройден, вы можете добавить зеленый сервер в балансировщик нагрузки. Постепенно отправляйте трафик через зеленый сервер, используя канареечное развертывание, пока не убедитесь, что все запросы выполнены успешно. Затем удалите исходный сервер и очередь.

При желании вы можете пропустить этап создания зеленого сервера и отправлять трафик с исходного сервера на SSL. Однако этот подход может помешать вам запустить канареечное развертывание. Вы обновляете конфигурацию сервера для прямого взаимодействия с новой очередью.

Упражнение 10.1

Какой порядок и группировку ресурсов вы бы использовали для рефакторинга и дробления монолита после анализа следующего кода:

```
if __name__ == "__main__":
    zones = ['us-west1-a', 'us-west1-b', 'us-west1-c']
    project.build()
    network.build(project)
    for zone in zones:
        subnet.build(project, network, zone)
    database.build(project, network)
    for zone in zones:
        server.build(project, network, zone)
    load_balancer.build(project, network)
    dns.build()
```

- a** DNS, балансировщик нагрузки, серверы, база данных, сеть + подсети, проект;
- b** балансировщик нагрузки + DNS, база данных, серверы, сеть + подсети, проект;
- c** проект, сеть + подсети, серверы, база данных, балансировщик нагрузки + DNS;
- d** база данных, балансировщик нагрузки + DNS, серверы, сеть + подсети, проект.

Ответ

Порядок и группировка, снижающие риск последствий рефакторинга, начинаются с DNS и продолжаются балансировщиком нагрузки, очередью, базой данных, серверами, сетью и подсетями, а также проектом (а). Вам следует начать с DNS как зависимости самого высокого уровня и развивать его отдельно от балансировщика нагрузки, поскольку балансировщик нагрузки требует атрибутов проекта и сети. Проведите рефакторинг ресурсов в следующем порядке: серверы, база данных, сеть и проект.

Рефакторинг серверов проводится раньше рефакторинга базы данных, поскольку серверы не управляют данными напрямую, а зависят от базы данных. Если у вас не получится провести рефакторинг базы данных, то вы хотя бы выделите серверы из монолита! Вы всегда можете оставить базу данных в монолите с сетью и проектом. Рефакторинг большей части ресурсов из монолита в достаточной степени облегчает масштабирование системы.

Упражнение 11.1

Команда сообщает, что ее приложение больше не может подключаться к другому приложению. Приложение работало на прошлой неделе, но с понедельника запросы не выполняются. Команда не вносила никаких изменений в свое приложение и подозревает, что проблема кроется в правиле брандмауэра. Какие шаги вы можете предпринять для устранения проблемы (выберите все подходящие варианты):

- a** войдите в панель управления аккаунтом облачного провайдера и проверьте правила брандмауэра для приложения;
- b** разверните новую инфраструктуру и приложения в «зеленой» среде для тестирования;
- c** изучите изменения в IaC для приложения;
- d** сравните правила брандмауэра у облачного провайдера с IaC;
- e** отредактируйте правила брандмауэра и разрешите весь трафик между приложениями.

Ответ

Это шаги a, c и d. Для устранения неполадок проверьте наличие дрейфа в правилах брандмауэра. Если вы не обнаружили отклонений, то можете поискать в IaC, на котором работает приложение, другие несоответствия. С точки зрения устранения неполадок эта проблема не обязательно требует новой среды для тестирования или разрешения всего трафика между приложениями.

Упражнение 12.1

Какие из приведенных ниже утверждений верны для данного кода (выберите все подходящие варианты):


```
HOURS_IN_MONTH = 730  
MONTHLY_BUDGET = 5000  
DATABASE_COST_PER_HOUR = 5  
NUM_DATABASES = 2  
BUFFER = 0.1
```

```
def test_monthly_budget_not_exceeded():  
    total = HOURS_IN_MONTH * NUM_DATABASES * DATABASE_COST_PER_HOUR  
    assert total < MONTHLY_BUDGET + MONTHLY_BUDGET * BUFFER
```

- a тест будет пройден, поскольку стоимость базы данных находится в пределах бюджета;
- b тест оценивает ежемесячную стоимость баз данных;
- c тест не учитывает различные типы баз данных;
- d в ходе теста вычисляется ежемесячная стоимость одного экземпляра базы данных;
- e тест включает в себя буфер в размере 10 % на случай превышения затрат в качестве мягкой политики.

Ответ

Правильные варианты – b, c и e. Общая ориентировочная стоимость баз данных в месяц равна 7300 долл., а ежемесячный бюджет с буфером 10 % составляет 5500 долл., поэтому тест не пройден. Сам тест не учитывает различные типы баз данных и не рассчитывает ежемесячную стоимость каждого экземпляра базы данных. Он берет за основу почасовой тариф и количество баз данных. Добавление буфера 10 % к ежемесячному бюджету несколько смягчает бюджетную политику. Небольшие изменения, которые приводят к незначительному превышению затрат, уменьшат трудности при доставке на производство, но послужат предупреждающим сигналом о возможном дальнейшем увеличении затрат.

Упражнение 12.2

Представьте, что у вас есть три сервера. Вы изучаете их использование и замечаете следующее:

- вам нужен один сервер для обслуживания минимального трафика;
- для обслуживания максимального трафика вам нужны три сервера;
- сервер обрабатывает трафик 24 часа в сутки, семь дней в неделю.

Что можно сделать, чтобы оптимизировать стоимость серверов на следующий месяц:

- a запланировать остановки ресурсов на выходные;
- b добавить `autoscaling_policy` для масштабирования в зависимости от памяти;

- c установить для всех серверов срок жизни три часа;
- d перенести серверы на машины с меньшим количеством процессоров и памяти;
- e перенести приложение в контейнеры и плотнее сжать их на серверах.

Ответ

Добавьте `autoscaling_policy` для масштабирования на основе памяти (b). Вы не можете запланировать полную остановку ресурсов на выходные, потому что вам нужен хотя бы один сервер на выходных. Установка срока действия или сокращение размеров серверов не поможет снизить затраты в данной ситуации. Перенос приложений в контейнеры предполагает долгосрочное решение, которое не может оптимизировать затраты на следующий месяц.

Предметный указатель

В

Blue-green развертывание, 301

Г

GitOps, 259

И

IaC, управляемая событиями, 443

А

Аварийное переключение, 295

Автомасштабирование, 405

Анализ

динамический, 187

статический, 186

Б

Бэкенд, 468

В

Ветка, 237

Внедрение зависимостей, 132

Внешняя проверка, 252

Воспроизводимость, 35

Входной контроль качества, 223

Выпуск, 175

Г

Гигиена кода, 74

Д

Дрейф конфигурации, 37, 67

Ж

Журнал изменений, 176

З

Зависимость

инфраструктурная, 118

однонаправленная, 119

Запрос на изменение, 241

Затраты

источник, 385

оценка, 391

И

Идемпотентность, 35
Избыточность, 293
Изменение
 срочное, 232
 стандартное, 232
Инверсия
 зависимостей, 125
 управления, 122
Инкрементное исправление, 374
Инструмент
 инициализации, 52
 создания образов, 54
 управления конфигурацией, 53
Инструментальное состояние, 130
Инфраструктура, 27, 29
 изменяемая, 62
 неизменяемая, 62
 хранящая состояние, 321

К

Канареечное развертывание, 308
Код как документация, 48
Компонуемость, 35
Конвейер поставки, 226
Контейнер как услуга, 28
Конфигурация
 «основной–основной», 299
 «основной–резервный», 294

Л

Линтинг, 76
Лицензия
 разрешительная, 428
 copyleft (антикопирайт), 428

М

Маскирование, 267
Менеджер конфигурирования, 130
Модель
 ветвления, 239
 SAMS, 33
Модуль, 92
Монолитная архитектура, 338

Н

Наблюдаемость системы, 292
Надежность, 295
Неизменность, 62
Непрерывная интеграция, 30, 224
Непрерывная поставка, 228
Непрерывное развертывание, 229

О

Обновление
 накат, 361
 откат, 361
Окно изменений, 44

П

Панель обзора изменений, 44
Парное программирование, 256
Паттерн
 «адаптер», 140
 «компоновщик», 93
 «медиатор», 146
 «одиночка», 89
 порождающий, 88
 проектирования ПО, 88
 «прототип», 100
 «строитель», 105
 структурный, 88
 фабричный, 97
 «фасад», 136
Переменная
 константная, 79
 конфиденциальная, 266
Пирамида тестирования, 215
Платформа как услуга (PaaS), 28
Поверхность атаки, 263
ПО как услуга (SaaS), 29
Политика, 262
 как код, 273
 механизм, 273
Последовательное обновление, 329
Принцип
 воспроизводимости, 35
 идемпотентности, 38
 компонуемости, 41
 наименьших привилегий, 264
 развиваемости, 42
Пробный прогон, 53, 196

Р

Радиус поражения, 30
Развиваемость, 35
Разработка
 магистрально ориентированная, 245
 функционально
 ориентированная, 240
Регрессионный тест, 314
Репозиторий
 единый, 159
 раздельный, 163
Ресурс
 верхнего уровня, 119
 вытесняемый (спотовый), 404
 нижнего уровня, 119
Рефакторинг, 328

С

Секрет, 84
Слияние, 225
Служба, 209
Среда тестирования модулей, 205
Срочное изменение, 65
Стиль IaC
 декларативный, 60
 императивный, 60
Стратегия тегирования, 269

Т

Тест
 интеграционный, 202

 контрактный, 199
 модульный, 190
 непрерывный, 213
 приемочный, 207
 регрессионный, 214
 сквозной, 209
Тестирование, 183
 в производстве, 418
 отрицательное, 279
 положительное, 279
Тестовая среда, 188
Тестовый стенд, 192, 463
Технический долг, 328

У

Управление
 версиями, 171
 семантическое, 172
 изменениями, 44, 223
 конфигурацией, 365

Ф

Файл README, 177
Фиксация версии, 171
Флаг функции (feature flag), 331
Функция как услуга (FaaS), 28

Э

Эталонная конфигурация
инфраструктуры, 61

Розмари Ванг

Инфраструктура как код

Главный редактор	<i>Мовчан Д. А.</i>
Зам. главного редактора	<i>Яценков В. С.</i> editor@dmkpress.com
Перевод	<i>Яценков В. С.</i>
Корректор	<i>Синяева Г. И.</i>
Верстка	<i>Чаннова А. А.</i>
Дизайн обложки	<i>Мовчан А. Г.</i>

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 39,33. Тираж 100 экз.

Веб-сайт издательства: **www.dmkpress.com**

Инфраструктура как код (IaC) — это набор методов и процессов для построения и обслуживания инфраструктуры с использованием скриптов, файлов конфигурации или языков программирования. IaC значительно облегчает тестирование компонентов, реализацию функций и масштабирование системы. Но самое главное — поскольку IaC следует классическим методам разработки, для внесения изменений в масштабах системы достаточно нескольких коммитов кода!

Прочитав эту книгу, вы научитесь создавать виртуальные серверы, сети, балансировщики нагрузки, брандмауэры и другие компоненты с помощью Python и Terraform, а также уверенно обновлять инфраструктуру, не прерывая работу системы.

Рассматриваемые темы:

- оптимизация модульной инфраструктуры и изоляция зависимостей;
- перенос систем в облако, устранение неполадок и смягчение их последствий;
- обновление инфраструктуры с минимальным временем простоя;
- использование паттернов для инструментов управления конфигурацией и сборки образов.

Книга адресована специалистам по инфраструктуре и профессиональным разработчикам, знакомым с языком Python, инструментами конфигурирования и публичными облачными сервисами.

Розмари Ванг — преподаватель, разработчик, писатель и лектор. Она работает над различными инфраструктурными проектами и принимает участие в разработке инструментов с открытым исходным кодом, таких как Terraform, Vault и Kubernetes.

«Глубокий разбор таких базовых понятий, как повторное использование, идемпотентность, модульность и развиваемость, при этом примеры не зависят от инструментов. Отличная книга!»

Люсиан Мали, Red Hat

«Понятное изложение, обширный охват темы. Вы узнаете, как автоматизировать свою инфраструктуру надежным и воспроизводимым способом».

Козимо Амманаси, ER Sistemi

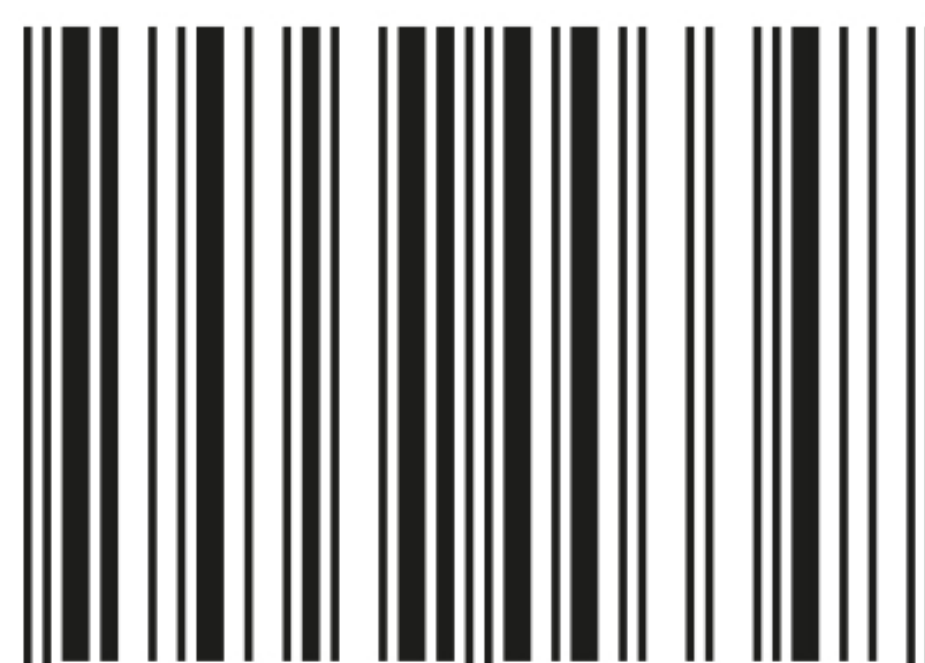
«Эта книга поможет командам применять передовые методы IaC в реальных проектах».

Дэвид Криф, Altansia

«Ключевые идеи и практические знания. Эта книга — отличный учебник!»

Джерemi Брайан, Anchore

ISBN 978-5-93700-334-8



9 785937 003348 >