Современная наука и инновации. 2025. № 3. С. 20-29.
ТЕХНИЧЕСКИЕ НАУКИ
ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

Modern Science and Innovations. 2025;(3):20-29..
TECHNICAL SCIENCE
INFORMATICS, COMPUTER ENGINEERING AND MANAGEMENT

Научная статья
УДК 004.75:004.415:004.9
https://doi.org/10.37493/2307-910X.2025.3.2

Легковесные конвейеры развертывания контейнерных сервисов на AWS: новый подход к устранению конфигурационного дрейфа в средах с высокой скоростью разработки

Сергей Андреевич Большаков[1*]

[1] Райффайзенбанк Чехия (Прага, Чехия)
[1] s.bolshakoff@outlook.com

*** Автор, ответственный за переписку**

Аннотация. Введение. *В современной практике DevOps существует фундаментальное противоречие между скоростью развертывания и согласованностью инфраструктуры. Конфигурационный дрейф, расхождение между декларируемым в коде состоянием и реальной конфигурацией облачных ресурсов, представляет собой критическую проблему для команд, использующих подход «Инфраструктура как код» (Infrastructure as Code, IaC).* ***Материалы и методы.*** *В статье предлагается минимальный, высокоскоростной конвейер развертывания для Amazon ECS, реализованный через GitHub Actions и опирающийся на новый атрибут track_latest в провайдере Terraform AWS.* ***Результаты и обсуждение****. В отличие от стандартного тяжеловесного цикла terraform plan + terraform apply, предложенная схема фокусируется на малой задержке и минимальной операционной сложности: Terraform остаётся источником правды для базовой инфраструктуры, а быстрые обновления образов делегируются лёгкому CI/CD-пути, что даёт практически мгновенные релизы без громоздких процедур. Такой подход особенно удобен для экспериментальных сред, MVP и ранних стадий продукта, где важны скорость итераций, простота процесса и сохранение детерминированной воспроизводимости изменений. Методологические преимущества предложенного подхода проявляются в институционализированной, формально специфицированной согласованности финального состояния инфраструктуры — что обеспечивает детерминированную репродуцируемость конфигураций — одновременно приводя к ощутимому сокращению как когнитивной, так и операционной нагрузки на команды развертывания; дополнительно архитектура повышает степень наблюдаемости за счёт глубокой нативной интеграции с системами трекинга задач и корпоративными каналами связи, позволяя агрегировать телеметрические сигналы, связывать события деплоя с артефактами жизненного цикла и оперативно выявлять отклонения.* ***Заключение.*** *В результате достигается прагматичный баланс между скоростью выпуска и конфигурационной инвариантностью, что превращает метод в эффективный инструмент для DevOps/SRE-инженеров, платформенных и продуктовых подразделений, облачных архитекторов и техлидов, стремящихся минимизировать drift и связанные с ним операционные риски в AWS.*

Ключевые слова: DevOps, CI/CD, Infrastructure as Code, Terraform, AWS, Amazon ECS, GitHub Actions, конфигурационный дрейф, непрерывное развертывание, автоматизация.

Для цитирования: *Большаков С. А. Легковесные конвейеры развертывания контейнерных сервисов на AWS: новый подход к устранению конфигурационного дрейфа в средах с высокой скоростью разработки // Современная наука и инновации. 2025. №3. С. 20-29. https://doi.org/10.37493/2307-910X.2025.3.2*

Research article

Lightweight deployment pipelines for containerized services on AWS: a new approach to eliminating configuration drift in high-velocity development environments

Sergey A. Bolshakov[1*]

[1] DevOps Lead, Raiffeisenbank Czech Republic (Prague, Czech Republic)
[1] s.bolshakoff@outlook.com

*** Corresponding author**

Abstract. Introduction. *A fundamental tension in the modern DevOps practice is between deployment velocity and infrastructure consistency. Configuration drift—that is, the divergence between the state declared in code and the actual configuration of cloud resources—represents an absolutely critical challenge for teams who manage Infrastructure as Code.* ***Materials and methods.*** *The present paper proposes a minimal, high-velocity deployment pipeline for Amazon ECS, implemented via GitHub Actions and leveraging the new track_latest attribute in the Terraform AWS provider.* ***Results and discussion.*** *Unlike the conventional, heavier terraform plan + terraform apply workflow, the proposed scheme emphasizes low latency and minimal operational complexity: Terraform remains the source of truth for core infrastructure, while frequent image updates are delegated to a lightweight CI/CD pipeline, thereby enabling near-instantaneous releases without cumbersome procedures. This approach is particularly well-suited to experimental environments, MVPs, and early-stage product development, where rapid iteration, process simplicity, and deterministic reproducibility of changes are paramount. The main methodological benefits of this approach are by way of formally specifying the infrastructure's final state consistency to ensure deterministic reproducibility of configurations while materially reducing both cognitive and operational load on deployment teams. Also, the setup boosts visibility with solid native hookups to bug trackers and company chat tools, allowing for the gathering of telemetry data linking deployment happenings to lifecycle items and quick spotting of any off-track moments.* ***Conclusion.*** *This gives a real-world mix between speed of release and steadiness of setup, making the method a strong tool for DevOps/SRE workers, platform and product groups, cloud planners, and tech leads wanting to cut drift and linked running dangers in AWS.*

Keywords: DevOps, CI/CD, Infrastructure as Code, Terraform, AWS, Amazon ECS, GitHub Actions, configuration drift, continuous deployment, automation.

For citation*: Bolshakov SA. Lightweight Containers Services Deployment Pipelines on AWS: A New Approach to Eliminate Configuration Drift in High-Speed Development Environments. Modern Science and Innovations. 2025;(3):20-29. (In Russ.). https://doi.org/10.37493/2307-910X.2025.3.2*

Введение. Эволюция практик DevOps за последнее десятилетие ознаменовала переход от базовых скриптов автоматизации к сложным, облачно-нативным конвейерам непрерывной интеграции и развертывания (CI/CD) [1]. Основной целью DevOps является сокращение жизненного цикла разработки и увеличение частоты выпуска релизов, что позволяет организациям быстрее реагировать на изменения рынка и потребности клиентов. Этот процесс привел к возникновению фундаментального противоречия между скоростью разработки (необходимостью частых и быстрых релизов) и операционным контролем (потребностью в стабильной, предсказуемой и безопасной инфраструктуре) [2].

Данное противоречие формирует своего рода «трилемму автоматизации развертывания», в которой организациям приходится выбирать между скоростью, контролем и простотой. Приоритеты варьируются в зависимости от контекста: крупные предприятия, особенно в финансовом секторе, часто отдают предпочтение контролю, управлению и безопасности, принимая на себя более высокие операционные издержки. Процесс выстраивается стартапами и командами, создающими минимально

жизнеспособные продукты, которые отдают приоритет крайне кратким итерациям для быстрого получения отзывов пользователей, что позволяет ускорить соответствие продукта рынку [3]. Инструмент управления релизами должен быть сложным, но не слишком сложным, чтобы дополнительная детализация контроля не снижала оперативности. С точки зрения скорости и простоты внедрения более простые схемы выигрывают, но, поскольку они не позволяют достаточно эффективно управлять конфигурациями, со временем накапливается дрейф конфигураций. Настоящее исследование сосредоточено на решении этой трилеммы путем предложения подхода, который обеспечивает все три аспекта: скорость, контроль и простоту.

Подход «Инфраструктура как код» (IaC) стал доминирующей парадигмой для управления современной облачной инфраструктурой, обеспечивая автоматизацию, воспроизводимость и версионирование. Однако его эффективность напрямую зависит от поддержания соответствия между кодом и реальным состоянием инфраструктуры. Конфигурационный дрейф (configuration drift) определяется как расхождение между декларируемым состоянием инфраструктуры в IaC-определениях (например, в коде Terraform) и ее фактическим состоянием в облачной среде [4]. Это нарушает основные принципы «Инфраструктуры как кода»: воспроизводимость должна быть гарантирована, но на самом деле она лишь обеспечивает возможность детального аудита эволюции конфигураций. Источников множество: любые ручные изменения через консоль облачного провайдера, внешние инструменты автоматизации, работающие в фоновом режиме, но не интегрированные с IaC, и даже неудачные или частично завершённые развёртывания — всё это может быть причиной [5]. Отсутствие контроля над их накоплением — это опасно: от простых ошибок в операциях и архитектуре, которые могут привести к появлению уязвимостей безопасности (например, открытые интерфейсы администрирования или неправильно настроенная модель прав доступа), до фактических нарушений требований (GDPR, HIPAA) и снижения стабильности работы, выражающегося как в частых сбоях развёртывания, так и в резком падении производительности.

Целью данной работы является введение и валидация нового, легковесного метода развертывания, который разрешает дилемму «скорость против контроля» для распространенного сценария — обновления контейнеризированных бэкенд-приложений на платформе AWS. Предлагаемое решение направлено на обеспечение скорости, характерной для прямых CLI-развертываний, при полном устранении связанного с ними риска конфигурационного дрейфа, тем самым сохраняя целостность принципов IaC.

Материалы и методы. Исследование легковесных конвейеров развертывания контейнерных сервисов на AWS опирается на анализ 11 источников, включающих академические статьи, отраслевые руководства, документацию провайдеров и практические кейсы внедрения IaC-решений. Теоретическую основу составили работы, рассматривающие эволюцию DevOps-практик и их влияние на скорость и надежность доставки программного обеспечения: Jain [1] показал, что интеграция DevOps и AI-подходов позволяет ускорять жизненный цикл разработки, а Jha и др. [3] раскрыли культурные аспекты DevOps, влияющие на восприятие командой гибких конвейеров. Ключевую роль играет также исследование Mercy [4], в котором конфигурационный дрейф был описан как системная угроза воспроизводимости инфраструктуры.

Методологически работа опирается на три взаимодополняющих направления анализа. Первое — систематический обзор практик управления конфигурацией и предотвращения дрейфа, включающий сопоставление определений и подходов (например, трактовка US Cloud [5] и рекомендации Firefly [8]) и анализ встроенных механизмов Terraform Enterprise [6] и Atlantis [7]. Второе направление — сравнительное исследование философии управления состоянием в IaC-инструментах: Pulumi [9, 10] и Google Cloud Run [11] рассматриваются как альтернативные модели, основанные на активном обнаружении

и исправлении дрейфа, в отличие от пассивного примирения изменений, предложенного в Terraform AWS Provider через атрибут track_latest.

Третье направление методологии связано с практическим моделированием конвейеров. В качестве исследовательского материала использовались сценарии развертывания контейнерных сервисов Amazon ECS, где традиционно применялись три альтернативных метода: тяжеловесные системы управления (Terraform Cloud/Enterprise, Atlantis), упрощенные CLI-подходы и новый гибридный паттерн, интегрирующий Terraform с GitHub Actions. Для каждого сценария была проведена оценка по ключевым критериям — скорость, риск конфигурационного дрейфа, операционная сложность, стоимость и управляемость — с опорой на опубликованные спецификации HashiCorp [6], отраслевые руководства Stadil [7] и документацию AWS.

Результаты и обсуждение. Корпоративные, «тяжеловесные» платформы проектируются для крупных организаций, где на передний план выходят требования к управлению, аудиту и жёсткой регламентации изменений инфраструктуры; в качестве иллюстрации можно привести Terraform Cloud/Enterprise от HashiCorp — управляемый сервис с централизованным стореджем состояния, коллективными workflow и механизмом принудительного соблюдения политик через Sentinel, — чья ценовая модель, основанная на RUM (ресурсы в час), делает его финансово неоптимальным для стартапов и небольших команд [6]. Аналогично, Atlantis — популярное OSS-решение, автоматизирующее запуск Terraform-команд через комментарии в pull-реквестах — при всей своей распространённости сталкивается с архитектурными ограничениями при масштабировании: его дефолтная однопоточная модель исполнения формирует бутылочные горлышки в сценариях множественных параллельных изменений, а self-hosted эксплуатация налагает скрытые операционные издержки на развёртывание, сопровождение, обеспечение безопасности и обновления; при этом ему недостаёт встроенных корпоративных возможностей уровня гранулярного RBAC и нативного обнаружения конфигурационного дрейфа [7].

Стремясь обойти сложность тяжеловесных платформ, команды обычно упрощают свой инструментарий до минимума, тем самым фрагментируя «источники истины» для инфраструктуры и увеличивая вероятность рассинхронизации состояний. Наиболее популярной реализацией такого упрощения является вызов AWS CLI непосредственно из конвейера CI/CD (например, aws ecs update-service для обновления task_definition), что приводит к тому, что ARN, фактически хранящийся в terraform-state для ресурса aws_ecs_service, больше не соответствует фактическому объекту в облаке. При последующем запуске плана terraform это несоответствие распознаётся, и Terraform пытается привести ресурс в соответствие с конфигурацией — декларативной конфигурацией, которая приводит к несоответствию ресурсов, возможному конфликту процессов и операционному риску. Другой популярный обход — хранение JSON-шаблона task definition прямо в репозитории приложения: это облегчает версионирование в едином репозитории кода, но нарушает принцип единого источника истины, распыляя инфраструктурную конфигурацию по нескольким артефактам. Тоже уязвим к ошибкам подход с terraform apply --refresh-only: он требует дисциплины инженеров, легко порождает состояния гонки и человеческие ошибки при ручном исполнении.

Предлагаемая парадигма опирается на эволюцию модели управления состоянием Terraform, позволяющую корректно «примирять» внешние изменения, не интерпретируя их автоматически как дрейф. В центре решения — атрибут track_latest, введённый для ресурса aws_ecs_service в Terraform AWS Provider (версия 5.37.0, февраль 2024), который трансформирует поведение управления конкретным поля: в отличие от lifecycle { ignore_changes = [task_definition] }, кардинально исключающего атрибут из контроля Terraform, присвоение track_latest = true инструктирует провайдер воздержаться от планирования изменений для task_definition, если оно было произведено извне, но при

одновременном применении других модификаций читать актуальное значение из облака и пассивно синхронизировать локальное состояние. Такой подход позволяет поддерживать согласованность terraform-стейта с деплоями, инициируемыми внешними инструментами (например, AWS CLI в GitHub Actions), избавляя от сложных и ненадёжных ухищрений с data-источниками и вычислениями через max() для определения последней ревизии, и по сути переводит Terraform из роли жёсткого «администратора» декларативного источника в более гибкий инструмент, способный к безопасной коэволюции с внешними процессами. Это знаменует собой зрелость инструментов IaC, признающих, что они функционируют в сложных экосистемах, а не в вакууме.

На рисунке 1 представлен пример конвейера GitHub Actions, реализующего предложенный механизм развертывания. Каждый шаг подробно аннотирован для ясности.

```yaml
name: Deploy
on:
  workflow_dispatch:
    inputs:
      environment:
        type: environment
        description: 'Environment'
jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    permissions:
      id-token: write # Необходимо для OIDC-аутентификации в AWS
      contents: read # Необходимо для checkout
    environment: ${{ github.event.inputs.environment }}
    steps:
      # 1. Подготовка тега образа на основе версии релиза
      - name: Prepare image tag
        id: prepare-image-tag
        run: |
          IMAGE_TAG=$(echo $GITHUB_REF_NAME | sed 's/^v//')
          echo "IMAGE_TAG=$IMAGE_TAG" >> $GITHUB_ENV

      # 2. Аутентификация в AWS с использованием OIDC
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-region: ${{ vars.AWS_REGION }}
          role-to-assume: arn:aws:iam::${{ vars.AWS_ACCOUNT_ID }}:role/${{ vars.CICD_ROLE_NAME }}
          role-session-name: GitHubWorkflowCI

      # 3. Загрузка текущего определения задачи ECS в качестве шаблона
      - name: Download current task definition
        run: |
          aws ecs describe-task-definition --task-definition ${{ vars.ECS_TASK_DEFINITION_NAME }} --query
taskDefinition > td.json

      # 4. Обновление тега образа в шаблоне определения задачи
      - name: Update the image
        id: update-image
        uses: aws-actions/amazon-ecs-render-task-definition@v1
        with:
          task-definition: td.json
          container-name: ${{ vars.ECS_CONTAINER_NAME }}
          image: ${{ vars.ECR_REPOSITORY_URL }}:${{ env.IMAGE_TAG }}

      # 5. Развертывание новой ревизии определения задачи и обновление сервиса ECS
      - name: Deploy
        uses: aws-actions/amazon-ecs-deploy-task-definition@v1
        with:
          task-definition: ${{ steps.update-image.outputs.task-definition }}
          service: ${{ vars.ECS_SERVICE_NAME }}
          cluster: ${{ vars.ECS_CLUSTER_NAME }}

      # 6. Ожидание стабилизации сервиса после развертывания
      - name: Wait for deployment to finish
        run: |
          aws ecs wait services-stable --cluster ${{ vars.ECS_CLUSTER_NAME }} --services ${{
vars.ECS_SERVICE_NAME }}
```

Рисунок 1. Пример конвейера GitHub Actions, реализующего предложенный механизм развертывания
Figure 1. An example GitHub Actions pipeline implementing the proposed deployment mechanism.

Соответствующая конфигурация Terraform для сервиса ECS становится предельно простой. Ключевым элементом является атрибут track_latest = true, как показано на рисунке 2.

```
resource "aws_ecs_service" "main" {
  name            = "my-app-service"
  cluster         = aws_ecs_cluster.main.id
  task_definition = aws_ecs_task_definition.main.arn
  desired_count   = 2
  track_latest    = true # Включение пассивного отслеживания

  #... другие параметры, такие как load_balancer, network_configuration и т.д.
}

resource "aws_ecs_task_definition" "main" {
  family                = "my-app"
  container_definitions = file("task-definition.json")
  #... другие параметры
}
```

Рисунок 2. Конфигурация Terraform для сервиса ECS
Figure 2. Terraform configuration for the ECS service

Этот подход позволяет управлять жизненным циклом основной инфраструктуры (сервис, кластер, балансировщик нагрузки) через Terraform, в то время как частые обновления, связанные с кодом приложения (версия образа в task_definition), делегируются легковесному и быстрому конвейеру CI/CD.

Предложенный метод систематически оценивается по ключевым критериям в сравнении с тяжеловесными и наивными легковесными подходами. Результаты анализа сведены в табл. 1.

Таблица 1. Сравнительная матрица подходов к развертыванию
Table 1. Comparative matrix of deployment approaches

Критерий	**Тяжеловесный подход (Terraform Cloud, Atlantis)**	**Легковесный подход (AWS CLI)**	**Предлагаемый подход (track_latest + GitHub Actions)**
Скорость развертывания	Средняя/Низкая	Высокая	Высокая
Риск конфигурационного дрейфа	Отсутствует	Высокий	Отсутствует
Операционная сложность	Высокая	Низкая	Низкая
Стоимость	Высокая (TFC) / Средняя (Atlantis)	Низкая	Низкая
Управляемость и аудит	Высокая	Низкая	Средняя (через историю Git)
Потенциал интеграции	Средний	Высокий	Высокий

Эта парадигма представляет собой синтез на самом базовом уровне характеристик этих двух классов, сохраняя простоту эксплуатации и минимальную сложность инструментов, характерные для лёгких решений, при этом сохраняя ту же надёжность и

устойчивость к дрейфу конфигурации, что и тяжёлые платформы. Таким образом, она представляется лучшим архитектурным вариантом в средах, где требуется очень тонкий компромисс между скоростью итераций и эксплуатационной стабильностью без существенных компромиссов. Другим важным преимуществом является глубокая и сквозная интеграция с существующим стеком инструментов DevOps, что упрощает адаптацию, снижает барьер внедрения и облегчает последующее масштабирование решений. Поскольку развертывание инициируется из репозитория приложения (например, при создании нового релиза в GitHub), создается бесшовный информационный поток, который повышает наблюдаемость для всех участников процесса. На Рисунке 3 представлена схема такой интеграции.

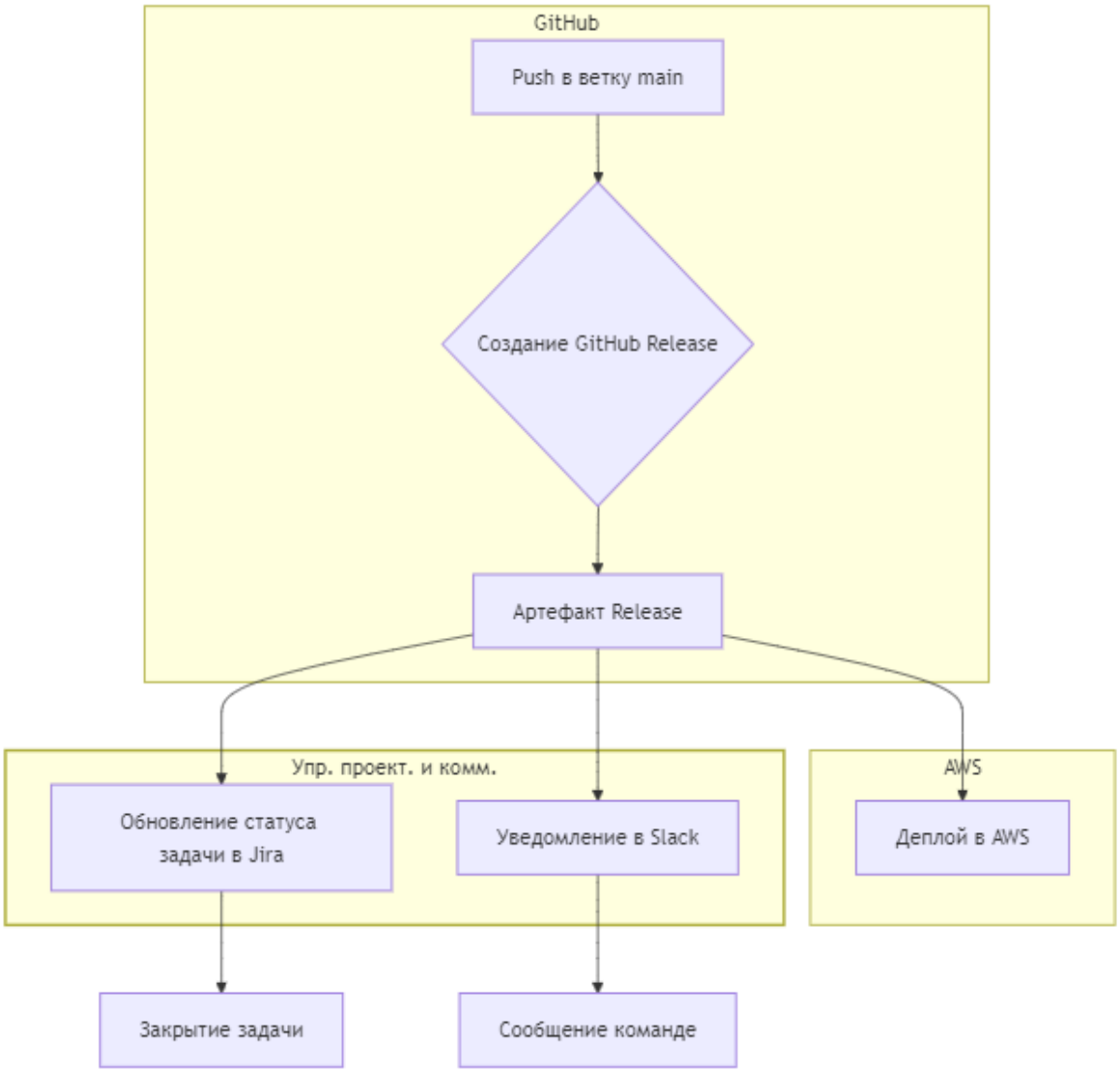

Рисунок 3. Схема интеграции конвейера развертывания с инструментами управления проектами

Figure 3. Integration diagram of the deployment pipeline with project management tools

Такая интеграция обеспечивает немедленную и автоматизированную видимость статуса развертывания для разработчиков, менеджеров проектов и QA-инженеров без необходимости ручного обновления статусов. Любое изменение в кодовой базе напрямую коррелирует с событием деплоя и соответствующей задачей в системе трекинга, что значительно упрощает аудит и отладку за счёт прозрачной привязки каждого артефакта к конкретному инциденту развертывания.

Для завершённого сравнения целесообразно сопоставить эту модель с парадигмами управления состоянием, принятыми в других IaC-инструментах и облачных платформах. Платформы вроде Pulumi и AWS CDK исповедуют проактивную стратегию детектирования дрейфа: команды pulumi refresh --preview-only и cdk drift служат для выявления расхождений между декларативной конфигурацией и «живым» состоянием с намерением их последующей корректировки, а не для пассивного согласования с внешними модификациями [9]. Кроме того, эти инструменты предоставляют механизмы импорта уже существующих ресурсов под своё управление — альтернативный путь интеграции «внешних» объектов в единый управляющий контур [10].

Подход Terraform с track_latest качественно выделяется на этом фоне: он концептуально легитимизирует внешние изменения как допустимую часть жизненного цикла ресурса и позволяет провайдеру синхронно принимать облачную реальность в качестве исходного факта, не шантажируя инфраструктуру принудительным откатом. Аналогично, на уровне платформ существуют решения (например, Google Cloud Run) с нативной моделью «ревизий», где управление трафиком между ревизиями реализовано как встроенный, программно управляемый механизм [11]. Хотя примитив развертывания схож, ключевое отличие предложенного в данной работе решения заключается в том, как именно инструмент IaC (Terraform) согласовывает свое состояние с этим внешним процессом. На Рисунке 4 представлено визуальное сравнение различных философий управления состоянием.

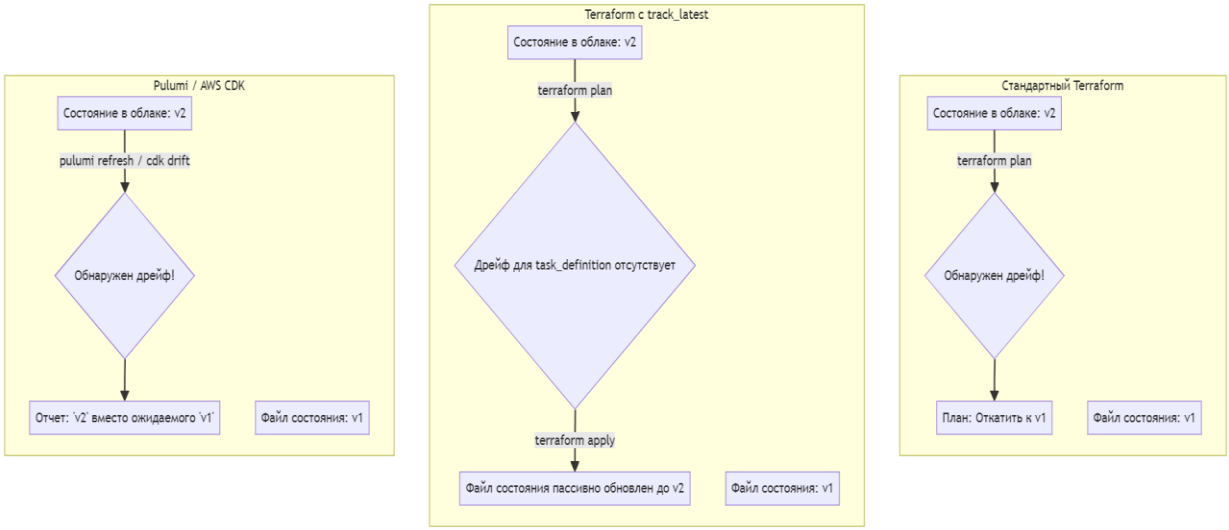

Рисунок 4. Сравнение философий управления состоянием IaC
Figure 4. Comparison of IaC state management philosophies

Эта схема наглядно демонстрирует уникальность подхода с track_latest, который не рассматривает внешнее обновление как ошибку, требующую исправления, а принимает его как легитимное изменение, пассивно обновляя свое внутреннее представление о состоянии инфраструктуры.

Заключение. В настоящей работе был представлен и проанализирован новый подход к развертыванию контейнеризированных сервисов на платформе AWS. Внедрение атрибута track_latest в Terraform AWS Provider открывает возможность для создания легковесных, надежных и свободных от конфигурационного дрейфа конвейеров CI/CD.

Основные выводы исследования заключаются в следующем. Предложенный механизм эффективно разрешает противоречие между скоростью развертывания и контролем над инфраструктурой, что особенно актуально для команд, работающих в средах с высокой скоростью итераций, таких как стартапы и проекты на стадии MVP. Он сочетает в себе конвергентные преимущества «тяжелых» систем — их известную

надежность и дрейф конфигурации — с явными преимуществами лёгких, «наивных» методов, то есть скорость и минимальную когнитивную нагрузку при реализации, одновременно снижая предвзятость, присущую каждой парадигме. Таким образом, развёртывания инициируются непосредственно из репозитория приложений, что позволяет перенести поток управления на модель, где наблюдаемость, отслеживание событий и автоматическая организация информационных каналов являются основными характеристиками процесса, что делает его воспроизводимым и ускоряет обратную связь между изменениями кода и их эксплуатационным воздействием.

В качестве направлений для будущих исследований можно выделить количественное сравнение производительности предложенного конвейера с тяжеловесными альтернативами, исследование применимости данного паттерна к другим сервисам AWS, которые могли бы извлечь выгоду из аналогичной модели согласования состояния, а также анализ долгосрочных последствий внедрения гибридной философии управления состоянием для эволюции практик IaC.

ЛИТЕРАТУРА / REFERENCES

1. Jain S. Integrating Artificial Intelligence with DevOps: Enhancing continuous delivery, automation, and predictive analytics for high-performance software engineering. World Journal of Advanced Research and Reviews. 2023. Т. 17, № 3. С. 1025–1043. DOI: 10.30574/wjarr.2023.17.3.0087. [Текст]
2. Pittet S. Continuous integration vs. continuous delivery vs. continuous deployment [Электронный ресурс]. Atlassian. URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment (дата обращения: 23.07.2025).
3. Jha A. V. и др. From theory to practice: Understanding DevOps culture and mindset. Cogent Engineering. 2023. Т. 10, № 1. DOI: 10.1080/23311916.2023.2251758
4. Mercy O. IaC Drift Detection At Scale: Terraform's Role in Enterprise Observability. International Journal of Novel Research and Development. 2024. [Электронный ресурс]. URL: https://www.researchgate.net/publication/392833971_IAC_DRIFT_DETECTION_AT_SCALE_TERRAFORM%27S_ROLE_IN_ENTERPRISE_OBSERVABILITY (дата обращения: 26.07.2025).
5. US Cloud. Configuration Drift [Электронный ресурс]. URL: https://www.uscloud.com/microsoft-support-glossary/configuration-drift/ (дата обращения: 26.07.2025).
6. HashiCorp. Terraform Enterprise [Электронный ресурс]. URL: https://developer.hashicorp.com/terraform/enterprise (дата обращения: 28.07.2025).
7. Stadil S. A Practical Guide to Terraform Operations with Atlantis [Электронный ресурс]. Scalr Learning Center. 22.05.2025. URL: https://scalr.com/learning-center/unlocking-advanced-terraform-operations-with-atlantis-a-practical-guide/ (дата обращения: 28.07.2025).
8. Firefly. How to Identify and Remediate Cloud Configuration Drift [Электронный ресурс]. URL: https://www.firefly.ai/academy/how-to-identify-and-remediate-cloud-configuration-drift-and-implement-best-practices-for-prevention (дата обращения: 29.07.2025).
9. Pulumi. Drift detection [Электронный ресурс]. URL: https://www.pulumi.com/docs/pulumi-cloud/deployments/drift/ (дата обращения: 30.07.2025).
10. Pulumi. Importing resources [Электронный ресурс]. URL: https://www.pulumi.com/docs/iac/adopting-pulumi/import/ (дата обращения: 01.08.2025).
11. Google Cloud. Rollbacks, gradual rollouts, and traffic migration [Электронный ресурс]. URL: https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration (дата обращения: 02.08.2025).

ИНФОРМАЦИЯ ОБ АВТОРАХ

Большаков Сергей Андреевич, Руководитель DevOps, Райффайзенбанк Чехия Прага, Чехия, s.bolshakoff@outlook.com

Вклад авторов: все авторы внесли равный вклад в подготовку публикации.
Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Статья поступила в редакцию 01.08.2025;
одобрена после рецензирования 13.09.2025;

принята к публикации 01.10.2025

INFORMATION ABOUT THE AUTHORS

Sergey Bolshakov, DevOps Lead, Raiffeisenbank Czech Republic, Prague, Czech Republic, s.bolshakoff@outlook.com

Contribution of the authors: the authors contributed equally to this article.
Conflict of interest: the authors declare no conflicts of interests.

The article was submitted: 01.08.2025;
approved after reviewing: 13.09.2025;
accepted for publication: 01.10.2025.