# Архитектурная Трансформация: Неделя 18, День 5 — Теория Непрерывного Развертывания (Continuous Deployment)

## 1. Введение: Эволюция парадигмы доставки программного обеспечения

### 1.1. От "Золотого Мастера" к Потоку Ценности

Пятый день восемнадцатой недели обучения знаменует собой критический переход в профессиональном сознании разработчика: сдвиг от создания артефактов к управлению живыми системами. В традиционной разработке игр (GameDev), особенно в экосистеме Unity, процесс релиза исторически воспринимался как дискретное, "точечное" событие. Разработчик компилирует "Golden Master" (финальный билд), упаковывает ресурсы в монолитные архивы (APK, IPA, EXE) и передает их платформодержателю (Steam, AppStore, Google Play). Этот процесс статичен, итерации занимают дни или недели, а цена ошибки высока, так как исправление требует повторной загрузки всего клиента пользователем.1

В мире Enterprise.NET Backend и облачной инфраструктуры такой подход является не просто устаревшим, он — архитектурно неприемлем. Бэкенд-сервис — это не файл на диске пользователя, а процесс, обрабатывающий тысячи транзакций в секунду в режиме 24/7. Переход к методологиям *Continuous Deployment (CD)* требует восприятия "релиза" не как события, а как рутинного, автоматизированного процесса, который может происходить десятки раз в день. Здесь вступает в силу принцип "Publish or Perish" (Публикуй или умри), диктующий, что код, который не находится в продакшене, не приносит бизнес-ценности.

### 1.2. Императив "Zero Downtime"

Фундаментальным отличием серверной разработки от клиентской является требование к доступности. Если игровой клиент требует перезапуска для обновления, это приемлемо. Если банковский API или сервис авторизации останавливается на 5 минут для обновления, это влечет прямые финансовые потери и репутационные риски.

Теория пятого дня сосредоточена на обеспечении *Zero Downtime Deployment* (развертывания с нулевым простоем). Это накладывает строгие ограничения на то, как приложение управляет своими соединениями, как оно реагирует на сигналы операционной системы (SIGTERM/SIGKILL) и, что наиболее критично, как эволюционирует схема базы данных. Мы переходим от вопроса "Как скомпилировать код?" к вопросу "Как заменить двигатель летящего самолета, не пролив кофе пассажиров?".

| **Характеристика** | **Традиционный GameDev (Unity)** | **Enterprise.NET Backend (CD)** |
| --- | --- | --- |
| **Единица доставки** | Монолитный бинарный файл (GB) | Docker-образ (MB), Микросервис |
| **Частота релизов** | Раз в месяц/спринт | Несколько раз в день/час |
| **Время простоя** | Допустимо (Maintenance Mode) | Недопустимо (Zero Downtime SLA 99.99%) |
| **Инициатор** | Ручной процесс (Release Manager) | Автоматический триггер (Git Merge) |
| **Откат (Rollback)** | Сложный, требует перекачки данных | Мгновенный (переключение трафика) |

## 2. Управление Артефактами и Реестры Контейнеров

### 2.1. Container Registry как Единый Источник Правды

В экосистеме CI/CD, построенной на контейнерах, центральным элементом инфраструктуры становится *Container Registry* (Реестр Контейнеров). Это специализированное хранилище, поддерживающее спецификации OCI (Open Container Initiative), которое служит мостом между процессами интеграции (CI) и развертывания (CD).

Для разработчика, использующего GitHub Actions, естественным выбором является **GitHub Container Registry (GHCR)**. В отличие от Docker Hub, который является публичным реестром общего назначения, GHCR обеспечивает тесную интеграцию с системой контроля версий и управления доступом GitHub. Образ в GHCR — это не просто файл, это иерархическая структура слоев (layers), манифестов и метаданных, позволяющая эффективно управлять версиями.2

#### 2.1.1. Безопасность Цепочки Поставок (Supply Chain Security)

В 2025 году просто собрать и запушить образ недостаточно. Современные стандарты безопасности (например, SLSA) требуют доказательства происхождения артефакта (Provenance). Это защита от атак типа "Man-in-the-Middle", когда злоумышленник подменяет образ между этапом сборки и этапом деплоя.

GitHub Actions позволяет генерировать *Attestations* — криптографически подписанные метаданные, связывающие конкретный SHA-хеш образа с конкретным запуском workflow, коммитом и репозиторием. Использование экшена actions/attest-build-provenance добавляет этот слой верификации, делая инфраструктуру устойчивой к компрометации.4

### 2.2. Аутентификация: GITHUB\_TOKEN против PAT

Критической ошибкой начинающих DevOps-инженеров является использование Personal Access Tokens (PAT) для аутентификации в реестрах внутри CI пайплайнов. PAT привязан к личному аккаунту сотрудника. Если сотрудник увольняется или меняет пароль, весь продакшен пайплайн останавливается ("Bus Factor" = 1). Кроме того, PAT часто имеет избыточные права ("god mode").

Архитектурно верным решением является использование автоматического секрета GITHUB\_TOKEN.

Этот токен обладает уникальными свойствами:

1. **Эфемерность:** Он создается в момент запуска джоба и уничтожается сразу после его завершения. Его невозможно украсть и использовать позже.2
2. **Минимальные привилегии (Least Privilege):** Права токена настраиваются декларативно в YAML файле workflow. Для публикации пакетов нам необходим строго определенный набор прав:

YAML

permissions:  
 contents: read # Чтение кода репозитория  
 packages: write # Пуш образов в GHCR  
 attestations: write # Подпись артефактов (Provenance)  
 id-token: write # Для OIDC федерации (если используется)

**Глубокий инсайт:** Важно отметить ограничение GITHUB\_TOKEN. Действия, совершенные с его помощью (например, пуш тэга или коммита), *не триггерят* запуск других workflow.6 Это сделано намеренно для предотвращения бесконечных рекурсивных циклов (CI Loop), когда бот бесконечно запускает сам себя. Если ваша стратегия релиза подразумевает, что пуш тэга должен запустить процесс деплоя в другом репозитории, придется использовать GitHub Apps или PAT, но внутри одного репозитория ограничение GITHUB\_TOKEN является благом, заставляющим проектировать пайплайны более линейно и предсказуемо.

## 3. Стратегии Версионирования и Тэгирования Артефактов

### 3.1. Анатомия Тэга и Опасность latest

В Docker-среде тэг latest является источником множества архитектурных проблем. Вопреки интуитивному пониманию, latest не означает "самая последняя версия". Это просто мутабельная строка (указатель), которая по умолчанию присваивается образу, если тэг не указан явно.

**Проблема:** В кластерной среде (Kubernetes, Docker Swarm) или при использовании масштабирования (Scaling), разные узлы могут скачать разные версии образа, помеченные одним и тем же тэгом latest, если кэш на узлах обновился в разное время. Это приводит к состоянию, известному как "Configuration Drift" (Дрейф конфигурации), когда часть серверов работает на версии v1, а часть на v2, что вызывает непредсказуемое поведение и трудноотлаживаемые баги.8

**Правило:** В продакшен-среде допустимо использовать только *иммутабельные* (неизменяемые) тэги.

### 3.2. Детерминированное Тэгирование: Git SHA

Наиболее надежным способом связи кода и артефакта является использование полного или короткого хеша коммита (Commit SHA).

Пример: myapp:sha-860c190.

Этот подход гарантирует полную прослеживаемость (Traceability). Если мониторинг показывает ошибку в контейнере с версией sha-860c190, инженер может выполнить команду git checkout 860c190 и гарантированно получить тот исходный код, который работает на сервере. Это устраняет класс проблем "на моей машине работает", связанных с рассинхронизацией версий.10

### 3.3. Семантическое Версионирование (SemVer)

Для публичных релизов и взаимодействия между командами хеши неудобны. Здесь применяется SemVer (v1.0.0).

Автоматизация этого процесса через docker/metadata-action позволяет генерировать каскад тэгов при создании Git-тэга.

Если разработчик пушит тэг v1.2.3, пайплайн должен создать следующие образы:

1. myapp:1.2.3 — Строгая фиксация.
2. myapp:1.2 — Плавающий тэг (Minor), указывающий на последний патч в ветке 1.2.
3. myapp:1 — Плавающий тэг (Major), указывающий на самую свежую версию в ветке 1.x.

Такой подход позволяет потребителям образа (например, другим микросервисам) подписываться на обновления безопасности (используя myapp:1), не рискуя получить ломающие изменения (которые, согласно SemVer, требуют смены мажорной версии).12

### 3.4. Ночные Сборки (Nightly Builds) и Cron

В крупных проектах интеграционные тесты могут занимать часы. Запускать их на каждый коммит нецелесообразно. Для этого используется стратегия ночных сборок.

GitHub Actions поддерживает триггер schedule с синтаксисом POSIX cron.

YAML

on:  
 schedule:  
 - cron: '0 2 \* \* \*' # Запуск каждый день в 02:00 UTC

Для таких сборок используется специализированный паттерн тэгирования, включающий дату: nightly-YYYYMMDD.14 Это создает временную шкалу артефактов, позволяя QA-команде тестировать "срез" состояния разработки на конкретный день.

Важно отметить, что cron-триггеры в GitHub Actions могут иметь задержку при высокой нагрузке на сервис, поэтому они не подходят для задач, требующих точности до секунды.16

## 4. Оптимизация Производительности CD Пайплайна

### 4.1. Кэширование Слоев Docker (Layer Caching)

В непрерывной доставке время цикла (Cycle Time) — от коммита до деплоя — является ключевой метрикой. Сборка Docker-образов для.NET приложений может быть медленной из-за тяжелых операций dotnet restore и компиляции. Без кэширования каждая сборка будет скачивать гигабайты пакетов NuGet заново.

GitHub Actions предоставляет продвинутый механизм кэширования через API — драйвер type=gha.18

Это решение превосходит старые методы (сохранение/загрузка образов через docker load/save), так как работает на уровне отдельных слоев файловой системы Docker и интегрировано напрямую в инфраструктуру раннера.

#### 4.1.1. Стратегия mode=max для Multi-Stage сборок

Для.NET приложений, использующих многоэтапные (Multi-stage) Dockerfile, критически важно использовать параметр mode=max.

По умолчанию (mode=min) кэшируются только слои финального образа. В случае с.NET, финальный образ содержит только рантайм и скомпилированные DLL. Все тяжелые слои SDK, Nuget-пакеты и промежуточные файлы компиляции находятся в build стейдже, который отбрасывается.

Использование mode=max заставляет BuildKit кэшировать и промежуточные слои. Это позволяет при повторной сборке (если csproj не менялся) пропустить этап восстановления пакетов и компиляции неизмененных проектов, сокращая время сборки с минут до секунд.18

YAML

- name: Build and push  
 uses: docker/build-push-action@v6  
 with:  
 cache-from: type=gha  
 cache-to: type=gha,mode=max

### 4.2. Изоляция Кэша (Cache Scope)

В репозитории с активной разработкой в нескольких ветках общий кэш может стать проблемой. Зависимости, добавленные в экспериментальной ветке, могут "отравить" кэш или вытеснить полезные слои основной ветки.

Параметр scope позволяет изолировать пространства имен кэша. Рекомендуемая практика — использовать имя ветки как часть ключа scope, но разрешать чтение из кэша основной ветки для ускорения создания новых веток.20

YAML

cache-from: |  
 type=gha,scope=${{ github.ref\_name }}-image  
 type=gha,scope=main-image  
cache-to: type=gha,mode=max,scope=${{ github.ref\_name }}-image

Это обеспечивает баланс между изоляцией и переиспользованием ресурсов.

## 5. Архитектура Развертывания: Стратегии и Реализация

Выбор стратегии развертывания зависит от требований бизнеса к доступности (SLA) и терпимости к рискам.

### 5.1. Сравнительный анализ стратегий

| **Стратегия** | **Механизм** | **Преимущества** | **Недостатки** | **Применимость** |
| --- | --- | --- | --- | --- |
| **Recreate** (Пересоздание) | docker stop -> docker rm -> docker run | Простота. Гарантирует отсутствие конфликтов версий (работает только v2). | **Downtime**. Сервис недоступен в течение времени старта. Потеря активных соединений. | Dev/Test среды. Некритичные сервисы. |
| **Rolling Update** (Постепенное) | Замена инстансов по одному (v1 -> v2). | Zero Downtime. Экономия ресурсов (не нужно удваивать инфраструктуру). | Одновременно работают v1 и v2. Сложность отладки. Риск гонки данных. | Kubernetes (стандарт), Stateless микросервисы. |
| **Blue/Green** (Сине-Зеленое) | Развертывание v2 (Green) рядом с v1 (Blue). Переключение роутера. | Мгновенное переключение. Мгновенный откат (Rollback). Возможность тестирования на проде. | Требует удвоения ресурсов (x2 RAM/CPU). Сложность управления состоянием (БД). | Mission-critical системы. |

### 5.2. Детальная анатомия Blue/Green Deployment

Для учебного проекта и многих Enterprise-систем Blue/Green является предпочтительным выбором из-за безопасности релизов.

**Компоненты системы:**

1. **Blue Environment (Active):** Текущая версия приложения, обслуживающая 100% пользовательского трафика.
2. **Green Environment (Idle):** Новая версия приложения, развернутая параллельно. Она полностью инициализирована, подключена к базам данных, прогрела кэши, но не получает внешнего трафика.
3. **Router/Load Balancer:** Точка входа (Nginx, Traefik, AWS ALB), которая решает, куда направлять запросы.

**Алгоритм перехода (The Switch):**

1. **Deploy Green:** CI/CD пайплайн разворачивает Green-контейнеры.
2. **Health Check & Smoke Test:** Система ожидает сигнала готовности от Green (/health). Запускается набор быстрых тестов против Green-версии (через приватный порт или внутреннюю сеть), чтобы убедиться в работоспособности бизнес-логики.
3. **Cutover (Переключение):** Роутер обновляет конфигурацию, перенаправляя входящий трафик с Blue на Green.
   * *В Nginx:* Это делается через изменение директивы upstream и команду nginx -s reload. Данная команда выполняет "graceful reload": новые запросы идут на новые процессы (Green), старые процессы дослуживают текущие соединения (Blue) и завершаются.
4. **Monitoring:** В течение некоторого времени (Canary period) мониторинг следит за метриками (HTTP 500, Latency).
5. **Cleanup или Rollback:**
   * Если ошибки растут: Роутер переключается обратно на Blue.
   * Если все стабильно: Blue-окружение останавливается (или остается в качестве Staging для следующего релиза).21

## 6. Управление Состоянием: Проблема Базы Данных в Zero Downtime

Самый сложный аспект CD — это не код, а данные. Если приложение stateless, то база данных (stateful) одна для обеих версий (Blue и Green). Это создает риск конфликта схем.

### 6.1. Сценарий конфликта

Представим, что в версии v2 мы решили переименовать колонку Username в Email.

1. Пайплайн запускает миграцию БД. Колонка Username удаляется, создается Email.
2. Green (v2) запускается и работает корректно.
3. Но в этот момент Blue (v1) все еще обслуживает пользователей!
4. Blue пытается выполнить запрос SELECT Username..., получает ошибку базы данных (Column not found).
5. Результат: Моментальный отказ обслуживания (Downtime) для всех пользователей Blue-версии до момента переключения.

### 6.2. Паттерн "Expand and Contract" (Parallel Change)

Для решения этой проблемы применяется паттерн миграции в несколько этапов, обеспечивающий обратную совместимость.24

**Фаза 1: Expand (Расширение)**

* **Действие:** Добавляем новую колонку Email, *не удаляя* старую Username.
* **Код:** Деплоим версию приложения, которая пишет данные *в обе* колонки, но читает по-прежнему из старой (или новой, если данные уже есть).
* **Результат:** База данных поддерживает и старую (Blue), и новую (Green) версию кода.

**Фаза 2: Migrate (Миграция данных)**

* **Действие:** Фоновый скрипт или джоб копирует данные из Username в Email для всех старых записей.
* **Код:** Работает версия, использующая новую колонку.

**Фаза 3: Contract (Сжатие)**

* **Действие:** Удаляем старую колонку Username.
* **Условие:** Это делается *только* после того, как все инстансы приложения гарантированно обновлены до версии, которая не использует старую колонку. Обычно это выносится в отдельный релиз.24

Интеграция с EF Core:

В.NET это означает отказ от автоматического применения миграций при старте приложения (context.Database.Migrate()) в продакшене. Миграции должны генерироваться в виде SQL-скриптов (идемпотентных) и применяться отдельным шагом CD-пайплайна до обновления приложения, но с строгим соблюдением принципа аддитивности (только добавление, не удаление).24

## 7. Жизненный цикл процесса: Graceful Shutdown

Обеспечение Zero Downtime невозможно, если приложение "грубо" обрывает соединения при остановке.

### 7.1. Сигналы ОС и Оркестрация

Когда Docker (или Kubernetes) останавливает контейнер, он посылает процессу с PID 1 сигнал SIGTERM.

Это вежливая просьба: "Пожалуйста, закончи свои дела и выйди".

Если приложение не завершается за отведенный таймаут (по умолчанию 30 секунд), система посылает SIGKILL — принудительное завершение.

Проблема ENTRYPOINT:

Если в Dockerfile использовать инструкцию в формате Shell: ENTRYPOINT./start.sh, то запускается оболочка /bin/sh, которая становится PID 1. Она не пробрасывает сигналы дочернему процессу приложения.NET. В итоге приложение не узнает о том, что его останавливают, пока не придет SIGKILL.

Решение: Использовать Exec-формат: ENTRYPOINT ["dotnet", "MyApp.dll"]. Это запускает процесс напрямую.27

### 7.2. Реализация в ASP.NET Core (Kestrel)

Веб-сервер Kestrel имеет встроенную поддержку Graceful Shutdown. При получении SIGTERM (который мапится в IHostApplicationLifetime.ApplicationStopping):

1. Сервер прекращает принимать *новые* TCP-соединения.
2. Существующие запросы продолжают обрабатываться.
3. Сервер ожидает завершения активных запросов в течение ShutdownTimeout.

Конфигурация:

По умолчанию в старых версиях.NET таймаут был 5 секунд, что может быть недостаточно для долгих операций (например, генерации отчета). В.NET 8+ и современных шаблонах это время увеличено, но его рекомендуется настраивать явно:

C#

// Program.cs  
builder.Services.Configure<HostOptions>(options =>  
{  
 // Даем приложению 30 секунд на завершение текущих запросов  
 // перед тем как оно будет убито  
 options.ShutdownTimeout = TimeSpan.FromSeconds(30);   
});

Этот параметр должен быть синхронизирован с настройкой terminationGracePeriodSeconds в Kubernetes или таймаутом docker stop.28

## 8. Автоматизация и Observability

### 8.1. Статусные Бейджи (Badges)

Неотъемлемой частью культуры Continuous Delivery является прозрачность. Статус последнего билда и деплоя должен быть виден всей команде. GitHub Actions позволяет генерировать Markdown-бейджи, отражающие состояние workflow.

Эти бейджи встраиваются в README.md репозитория, создавая "приборную панель" здоровья проекта.

Синтаксис: !(https://github.com/USER/REPO/actions/workflows/main.yml/badge.svg).30

### 8.2. Интеграция с обратной связью

CD пайплайн не заканчивается на команде docker run. Он заканчивается верификацией.

Продвинутые пайплайны включают шаги:

* **Smoke Tests:** curl запрос к развернутому сервису с ожиданием кода 200. Используются циклы с таймаутом (wait-for-it скрипты) для ожидания поднятия сервиса.32
* **Notification:** Отправка уведомления в Slack/Teams/Discord о статусе деплоя (успех/провал) с ссылкой на коммит и логи.

## 9. Практическое задание: Capstone Project "Zero-Click Pipeline"

В рамках итогового проекта студенту предстоит реализовать полный цикл CD для сервиса уведомлений.

### 9.1. Архитектура Пайплайна

Файл .github/workflows/cd.yml должен реализовывать следующую логику:

1. **Триггеры:** Push в ветку main.
2. **Job 1: Test & Build:**
   * Запуск юнит-тестов.
   * Сборка Docker-образа с использованием кэша (type=gha,mode=max).
   * Генерация метаданных (SemVer, SHA).
   * Пуш в GHCR (с правами packages: write).
   * Генерация Provenance аттестации.
3. **Job 2: Deploy (Simulation):**
   * Зависит от успеха Job 1 (needs:).
   * Подключение к удаленному серверу (или локальному раннеру) через SSH.
   * Выполнение скрипта "Blue/Green" обновления через Docker Compose:
     + docker pull нового образа.
     + Поднятие нового сервиса на свободном порту.
     + Проверка здоровья (curl).
     + Переключение Nginx конфига (шаблонизация через envsubst).
     + Перезагрузка Nginx.
     + Остановка старого контейнера.

Этот проект консолидирует все знания недели: от конфигурации и Dockerfile до управления сетями и автоматизации, завершая трансформацию Unity-разработчика в инженера, способного проектировать и поддерживать надежные серверные системы.

#### Источники

1. План обучения CI/CD и конфигурации .NET неделя 13
2. Working with the Container registry - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/packages/working-with-a-github-packages-registry/working-with-the-container-registry>
3. GitHub Container Registry - Qovery, дата последнего обращения: декабря 6, 2025, <https://www.qovery.com/docs/configuration/integrations/container-registries/github-cr>
4. Add SBOM and provenance attestations with GitHub Actions - Docker Docs, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/build/ci/github-actions/attestations/>
5. Using artifact attestations to establish provenance for builds - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/actions/security-for-github-actions/using-artifact-attestations/using-artifact-attestations-to-establish-provenance-for-builds>
6. GITHUB\_TOKEN - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/en/actions/concepts/security/github_token>
7. Triggering a workflow - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/actions/using-workflows/triggering-a-workflow>
8. Properly Versioning Docker Images - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/56212495/properly-versioning-docker-images>
9. Docker Tagging: Best practices for tagging and versioning docker images - Steve Lasker, дата последнего обращения: декабря 6, 2025, <https://stevelasker.blog/2018/03/01/docker-tagging-best-practices-for-tagging-and-versioning-docker-images/>
10. What are best practices for docker tag versioning? - DevOps Stack Exchange, дата последнего обращения: декабря 6, 2025, <https://devops.stackexchange.com/questions/1508/what-are-best-practices-for-docker-tag-versioning>
11. GitHub Action to extract metadata (tags, labels) from Git reference and GitHub events for Docker, дата последнего обращения: декабря 6, 2025, <https://github.com/docker/metadata-action>
12. Manage tags and labels with GitHub Actions - Docker Docs, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/build/ci/github-actions/manage-tags-labels/>
13. Publishing Semantic Versioned Docker Images to GitHub Packages Using GitHub Actions, Featuring Attestations | by Jared Hatfield | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@jaredhatfield/publishing-semantic-versioned-docker-images-to-github-packages-using-github-actions-ebe88fa74522>
14. Docker Metadata action - GitHub Marketplace, дата последнего обращения: декабря 6, 2025, <https://github.com/marketplace/actions/docker-metadata-action>
15. GitHub Action to extract metadata (tags, labels) for Docker, дата последнего обращения: декабря 6, 2025, <https://github.com/mithro/docker-metadata-action>
16. Events that trigger workflows - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/actions/learn-github-actions/events-that-trigger-workflows>
17. Run your GitHub Actions workflow on a schedule - Jason Etcovitch, дата последнего обращения: декабря 6, 2025, <https://jasonet.co/posts/scheduled-actions/>
18. GitHub Actions cache - Docker Docs, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/build/cache/backends/gha/>
19. docker cache comparison · Issue #377 · nakamasato/github-actions-practice, дата последнего обращения: декабря 6, 2025, <https://github.com/nakamasato/github-actions-practice/issues/377>
20. GitHub build-push-action and gha cache, should I set the github.ref\_name in the scope?, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/76227392/github-build-push-action-and-gha-cache-should-i-set-the-github-ref-name-in-the>
21. 5 Blue-Green Deployment Best Practices for Zero-Downtime Releases - Coherence, дата последнего обращения: декабря 6, 2025, <https://www.withcoherence.com/articles/5-blue-green-deployment-best-practices-for-zero-downtime-releases>
22. How I Set Up a Resilient Blue/Green Deployment with Docker Compose and Nginx, дата последнего обращения: декабря 6, 2025, <https://medium.com/@ChinecheremU/how-i-set-up-a-resilient-blue-green-deployment-with-docker-compose-and-nginx-9726a9a068bf>
23. Zero-downtime Deployments with Docker Compose & Nginx - Immersed in Code, дата последнего обращения: декабря 6, 2025, <https://immersedincode.io.vn/blog/zero-downtime-deployment-with-docker-compose-nginx/>
24. Applying Migrations - EF Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/applying>
25. Expand and Contract - A Pattern to Apply Breaking Changes to Persistent Data with Zero Downtime - Tim Wellhausen, дата последнего обращения: декабря 6, 2025, <https://www.tim-wellhausen.de/papers/ExpandAndContract/ExpandAndContract.html>
26. The Difference Between Rolling and Blue-Green Deployments | Blog - Harness, дата последнего обращения: декабря 6, 2025, <https://www.harness.io/blog/difference-between-rolling-and-blue-green-deployments>
27. How to send SIGTERM (graceful shutdown) to a .NET Core process in MacOS/Linux, дата последнего обращения: декабря 6, 2025, <https://blog.steadycoding.com/how-to-send-sigterm-and-more-to-a-net-core-process/>
28. Real Graceful Shutdown in Kubernetes and ASP.NET Core - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/arminshoeibi/real-graceful-shutdown-in-kubernetes-and-aspnet-core-2290>
29. How do I configure Kestrel to ignore shutdown (CancellationToken/SIGTERM), дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/79507120/how-do-i-configure-kestrel-to-ignore-shutdown-cancellationtoken-sigterm>
30. Adding a workflow status badge - GitHub Docs, дата последнего обращения: декабря 6, 2025, <https://docs.github.com/actions/managing-workflow-runs/adding-a-workflow-status-badge>
31. GitHub Markdown Badges Explained - Daily.dev, дата последнего обращения: декабря 6, 2025, <https://daily.dev/blog/github-markdown-badges-explained>
32. Actions · GitHub Marketplace - Wait For HTTP Responses, дата последнего обращения: декабря 6, 2025, <https://github.com/marketplace/actions/wait-for-http-responses>
33. Wait for an HTTP endpoint to return 200 OK with Bash and curl - gist/GitHub, дата последнего обращения: декабря 6, 2025, <https://gist.github.com/rgl/f90ff293d56dbb0a1e0f7e7e89a81f42>