# Архитектурная Трансформация и Паттерны Конфигурации в Enterprise.NET: Теоретический Базис Недели 18

## 1. Эпистемологический Сдвиг: От Монолитных Клиентов к Распределенным Системам

### 1.1. Фундаментальная дихотомия: Stateful vs. Stateless

Переход от разработки клиентских приложений на игровых движках, таких как Unity, к инженерной разработке серверных решений корпоративного уровня на платформе.NET Core знаменует собой не просто смену инструментария, но фундаментальную трансформацию ментальной модели инженера. Восемнадцатая неделя обучения, согласно утвержденному плану 1, является критической точкой бифуркации, где фокус смещается с императивного управления состоянием внутри игрового цикла на декларативное управление конфигурацией в распределенной среде.

В экосистеме GameDev архитектура традиционно строится вокруг сохранения состояния (Stateful). Игровой клиент — это монолитная сущность, где данные часто жестко "запекаются" в бинарные файлы, ScriptableObjects или префабы, становясь неотъемлемой частью дистрибутива.1 Жизненный цикл таких настроек неразрывно связан с жизненным циклом сборки: изменение скорости персонажа или баланса оружия требует перекомпиляции и обновления клиента через магазины приложений. Конфигурация здесь рассматривается как статический актив (Asset), который загружается в память при старте и существует в единственном экземпляре, часто доступном через глобальный синглтон типа GameManager.Instance.4

В мире Enterprise.NET бэкенд-сервис существует в принципиально иной, враждебной и динамичной среде. Приложение должно быть **stateless** (без сохранения состояния) и **environment-agnostic** (независимым от среды).5 Это требование диктуется методологией **The Twelve-Factor App**, которая постулирует строгое разделение кода и конфигурации. Один и тот же Docker-образ, собранный единожды, должен проходить через все стадии конвейера (Dev -> QA -> Staging -> Production), изменяя свое поведение исключительно за счет внешней инъекции конфигурации.1 В отличие от игрового мира, где "окружение" — это сцена Unity, в бэкенде окружение — это эфемерная инфраструктура, управляемая оркестраторами вроде Kubernetes, где инстансы сервисов могут создаваться и уничтожаться динамически.

### 1.2. Сравнительный анализ ментальных моделей

Для успешной адаптации Unity-разработчика к реалиям бэкенда необходимо провести деконструкцию привычных паттернов и сопоставить их с новыми архитектурными примитивами.NET.

| **Характеристика** | **Подход Unity (GameDev)** | **Подход Enterprise.NET (Backend)** | **Архитектурная Импликация** |
| --- | --- | --- | --- |
| **Источник правды** | ScriptableObject, JSON в StreamingAssets, PlayerPrefs. Данные упакованы внутри билда.1 | Иерархия провайдеров: Переменные окружения, CLI аргументы, Файлы (Secrets Mounts), Azure Key Vault.7 | Конфигурация внешняя по отношению к коду. Код не знает, откуда пришли данные. |
| **Изменяемость** | Изменения в Runtime часто не сохраняются (в редакторе) или требуют перезапуска. | Горячая перезагрузка (Hot Reload) через IOptionsSnapshot и IOptionsMonitor без даунтайма.8 | Сервис должен уметь перестраивать свое поведение "на лету" под нагрузкой. |
| **Доступ к данным** | Прямой доступ к полям статических классов или синглтонов (Manager.Instance.Value).9 | Внедрение зависимостей (DI) через интерфейсы IOptions<T>.8 | Инверсия контроля (IoC) и тестируемость. Отказ от глобального состояния. |
| **Безопасность** | Обфускация кода, кастомное шифрование сохранений. Секреты часто в коде. | Строгое разделение: User Secrets (Dev) и Environment/Vault (Prod). Принцип envdir.11 | Исключение секретов из системы контроля версий. Секреты монтируются в RAM-диск. |
| **Конфликты** | Merge conflict в YAML-файлах сцен и префабов. | Приоритет провайдеров конфигурации (Last Write Wins).12 | Детерминированный алгоритм перезаписи настроек слоями инфраструктуры. |

Этот сдвиг требует отказа от привычки "настроить в инспекторе и нажать Play". В.NET "инспектором" становится совокупность файлов appsettings.json, переменных окружения и секретных хранилищ, которые собираются в единое целое только в момент запуска приложения.

## 2. Анатомия Конфигурационной Подсистемы.NET

### 2.1. Концепция Провайдеров и Иерархия Слоев

В отличие от старых версий ASP.NET, где использовался монолитный Web.config, современный.NET Core использует модульную систему **послойной конфигурации** (Layered Configuration). Интерфейс IConfiguration — это фасад, скрывающий за собой агрегацию множества источников данных (провайдеров), которые загружаются в строго определенном порядке.13

Механизм WebApplication.CreateBuilder(args) инициализирует хост, который автоматически регистрирует дефолтный набор провайдеров. Понимание порядка их загрузки критически важно, так как каждый последующий провайдер имеет возможность переопределить значения, заданные предыдущими. Это реализует фундаментальный принцип **"Last Write Wins"** (Последний записавший побеждает).10

#### Детальный разбор приоритетов загрузки

Архитектура.NET выстраивает следующую цепочку ответственности для конфигурационных слоев (от низшего приоритета к высшему):

1. appsettings.json:  
   Это базовый слой, аналог "Default Prefab" в Unity. Файл содержит структурную схему конфигурации и значения по умолчанию, которые безопасны для использования в любой среде (например, настройки форматирования логов). Он загружается первым и формирует "скелет" конфигурационного объекта.7
2. appsettings.{Environment}.json:  
   Специализированный слой среды. Среда выполнения определяется системной переменной ASPNETCORE\_ENVIRONMENT (или DOTNET\_ENVIRONMENT). Если приложение запущено в режиме Development, загружается appsettings.Development.json. Значения из этого файла переопределяют (override) значения из базового JSON. Это архитектурный эквивалент Prefab Variant в Unity, где вариант наследует структуру, но модифицирует конкретные поля.13 Важно отметить, что в продакшене часто используется пустой или минималистичный appsettings.Production.json, так как реальные боевые настройки (строки подключения, ключи) не должны храниться в файлах репозитория.
3. User Secrets (Секреты пользователя):  
   Этот слой активен только в среде Development. Он представляет собой локальное JSON-хранилище, расположенное вне директории проекта (в профиле пользователя ОС). Это критически важный механизм защиты, предотвращающий случайный коммит чувствительных данных (паролей, токенов) в систему контроля версий (Git). Для Unity-разработчика это можно сравнить с локальными настройками редактора, которые игнорируются при коммите, но с гарантией физической изоляции данных.16
4. Переменные окружения (Environment Variables):  
   Слой инфраструктуры. Переменные, установленные операционной системой или контейнеризатором (Docker, Kubernetes), имеют приоритет над всеми файловыми источниками. Обычно провайдер настроен на чтение переменных с префиксами ASPNETCORE\_ или DOTNET\_.7 Это основной механизм конфигурации в облачных средах (Cloud Native), позволяющий изменять поведение сервиса без пересборки контейнера. Например, DevOps-инженер может переопределить строку подключения к базе данных, просто изменив YAML-манифест Kubernetes, и приложение подхватит это изменение при рестарте пода.
5. Аргументы командной строки (Command-line arguments):  
   Наивысший приоритет. Значения, переданные непосредственно при запуске процесса (например, dotnet run --urls="http://\*:5000"), переписывают абсолютно все предыдущие слои.7 Этот слой используется преимущественно для отладки, CI/CD скриптов или экстренных вмешательств, когда необходимо форсированно изменить параметр запуска без доступа к файловой системе или окружению контейнера.

### 2.2. Проблема Слияния Массивов (Array Merging)

Одним из наиболее коварных аспектов конфигурационной подсистемы.NET, вызывающим множество ошибок при переходе с других платформ, является механизм слияния массивов. В отличие от Unity, где переопределение списка в инспекторе варианта префаба обычно заменяет список целиком,.NET Configuration использует **поиндексное слияние**.18

Техническая деталь:

Конфигурация в.NET внутренне представлена как плоский словарь (Dictionary) пар "Ключ-Значение". Массив Items: транслируется в ключи:

* Items:0 = "A"
* Items:1 = "B"

Если в переопределяющем слое (например, в переменных окружения) вы попытаетесь "заменить" массив на ["C"], система увидит это как:

* Items:0 = "C"

При слиянии слоев происходит следующее:

1. Индекс 0 из базового слоя ("A") перезаписывается значением из нового слоя ("C").
2. Индекс 1 из базового слоя ("B") **остается**, так как в новом слое нет значения для индекса 1.

В результате, вместо ожидаемого ["C"], приложение получает гибридный массив ``. Это поведение является источником трудноуловимых багов.19 Решение этой проблемы требует либо явного "зануления" лишних индексов (что неудобно), либо использования кастомных биндеров, либо архитектурного отказа от использования массивов для конфигурации, которая должна полностью замещаться, в пользу словарей или примитивов, разделенных запятыми.18

### 2.3. Flattening и Разделители: Проблема Двоеточия

Иерархическая структура JSON (Section:SubSection:Key) естественна для.NET, но проблематична для некоторых операционных систем. Стандартный разделитель иерархии — двоеточие (:). Однако в Linux (и, следовательно, в Docker/Kubernetes) двоеточие часто является зарезервированным символом в оболочках (bash) или недопустимо в именах переменных окружения.13

Для решения этой проблемы в.NET внедрен механизм **автоматической трансляции двойного подчеркивания** (\_\_). Провайдер переменных окружения автоматически преобразует \_\_ в : при построении ключей конфигурации.

Пример:

Переменная окружения Database\_\_ConnectionSettings\_\_Host, установленная в Docker-контейнере, будет воспринята приложением как иерархический ключ Database:ConnectionSettings:Host.

Это позволяет передавать глубоко структурированную конфигурацию через плоские списки переменных окружения, что является стандартом де-факто в микросервисной архитектуре.13

## 3. Паттерн Options: Строгая Типизация и Управление Временем Жизни

### 3.1. Отказ от "Магических Строк"

Прямое использование интерфейса IConfiguration (например, вызов \_configuration) в бизнес-логике считается анти-паттерном в Enterprise-разработке. Такой подход:

1. Порождает "магические строки", разбросанные по всему коду.
2. Лишает разработчика преимуществ статической типизации и IntelliSense.
3. Делает рефакторинг конфигурации трудоемким и опасным процессом.
4. Затрудняет тестирование, так как требует мокирования сложного словаря конфигурации.

Решением является **Options Pattern** — механизм проецирования секций конфигурации на строго типизированные C# классы (POCO - Plain Old CLR Objects). Это позволяет работать с настройками как с объектами, инкапсулирующими связанные параметры.1

### 3.2. Интерфейсы Options и их Поведенческие Характеристики

Система внедрения зависимостей (DI) в ASP.NET Core предоставляет три основных интерфейса для потребления опций. Выбор правильного интерфейса критически важен для производительности и корректности работы приложения, так как они имеют принципиально разные жизненные циклы и механизмы обновления.8

#### IOptions (Singleton)

* **Регистрация:** services.Configure<MyOptions>(...)
* **Жизненный цикл:** Синглтон (Singleton). Объект опций создается один раз при первом обращении (lazy initialization) и кэшируется навсегда.
* **Поведение:** "Снимок при старте". Игнорирует любые изменения в источнике конфигурации (файлах JSON) после запуска приложения.
* **Use Case:** Идеально подходит для статических настроек, которые не должны меняться без перезапуска процесса (например, ключи шифрования AES, идентификаторы инстансов).
* **Производительность:** Максимальная. Нулевые аллокации при повторных обращениях, так как возвращается один и тот же экземпляр из кэша.24

#### IOptionsSnapshot (Scoped)

* **Жизненный цикл:** Scoped. Создается заново для каждого HTTP-запроса (или области видимости DI).
* **Поведение:** "Снимок на запрос". При каждом создании заново считывает конфигурацию из исходных провайдеров. Это обеспечивает функциональность **Hot Reload** — если вы измените appsettings.json на работающем сервере, следующий HTTP-запрос получит обновленные значения без рестарта приложения.
* **Внутреннее устройство:** Использует IOptionsFactory для пересчета значений на каждом запросе.
* **Архитектурное ограничение:** Не может быть внедрен в Singleton-сервисы. Попытка внедрить IOptionsSnapshot в конструктор синглтона приведет к ошибке времени выполнения (Captive Dependency), так как синглтон захватит scoped-объект, нарушая его жизненный цикл.25
* **Влияние на производительность (GC Pressure):** В высоконагруженных системах (High Load) массовое использование IOptionsSnapshot может создавать значительную нагрузку на сборщик мусора (GC). Поскольку объект создается заново на каждый запрос (даже если конфиг не менялся), это приводит к постоянным аллокациям памяти. В.NET 8 были проведены оптимизации, но архитектурная стоимость пересоздания объекта остается.8

#### IOptionsMonitor (Singleton)

* **Жизненный цикл:** Синглтон (Singleton).
* **Поведение:** Реактивный монитор. Предоставляет свойство CurrentValue (актуальное значение) и механизм подписки на изменения OnChange(Action<T>).
* **Внутреннее устройство:** Использует примитив IChangeToken для подписки на события файловой системы или других источников. Обновляет внутренний кэш *только* когда происходит реальное изменение конфигурации.
* **Use Case:** Идеален для Singleton-сервисов, которым требуется реактивность (например, изменение уровня логирования LogLevel на лету, ротация токенов доступа, управление Feature Flags).8
* **Преимущество:** Сочетает производительность IOptions (синглтон) с гибкостью IOptionsSnapshot (обновление), избегая накладных расходов на постоянное пересоздание объектов.28

### 3.3. Сравнительная Таблица Интерфейсов

Для закрепления понимания приведем сводную таблицу характеристик:

| **Характеристика** | **IOptions<T>** | **IOptionsSnapshot<T>** | **IOptionsMonitor<T>** |
| --- | --- | --- | --- |
| **Время жизни (Lifetime)** | Singleton | Scoped | Singleton |
| **Реакция на изменения** | Нет (Static) | Да (на след. запрос) | Да (мгновенно, Real-time) |
| **Инъекция в Singleton** | Допустима | **Запрещена** | Допустима |
| **Performance Overhead** | Низкий | Высокий (GC Allocations) | Низкий (Event-based) |
| **Аналогия с Unity** | ScriptableObject в билде | Чтение файла в Update() | ScriptableObject в Editor (OnValidate) |

### 3.4. Ментальная Модель для Unity Разработчика

Для разработчика из GameDev важно провести четкую параллель:

* ScriptableObject в Unity — это, по сути, IOptions<T>. Данные загружаются в память и существуют глобально. В скомпилированной игре они статичны.
* Использование IOptionsSnapshot архитектурно похоже на чтение файла с диска (File.ReadAllText) или десериализацию JSON каждый кадр в методе Update(). Это объясняет, почему злоупотребление Snapshot может "просадить FPS" (снизить пропускную способность - Throughput) веб-сервиса: вы платите процессорным временем за свежесть данных.1
* В Unity изменение данных в ScriptableObject во время игры (Runtime) часто не сохраняется или ведет к рассинхронизации. В.NET использование IOptionsMonitor позволяет корректно обрабатывать изменения состояния, уведомляя подписчиков через события, аналогично C# Events (Action), что позволяет системе перестроиться без перезагрузки.9

## 4. Стратегии Валидации и Обеспечения Целостности: Fail Fast

Одной из критических уязвимостей в разработке бэкенда является **"Configuration Drift"** или запуск приложения с некорректными настройками. В Unity, если вы забыли назначить префаб в инспекторе, вы увидите UnassignedReferenceException сразу при запуске игры. В бэкенде, если строка подключения к базе данных отсутствует или некорректна, приложение может успешно запуститься (Health Check пройдет), но начнет выдавать ошибки 500 пользователям при попытке выполнить запрос. Это недопустимо.

Приложение должно следовать принципу **Fail Fast** — падать мгновенно при старте, если его конфигурация невалидна. Это предотвращает деплой "сломанных" инстансов и сигнализирует оркестратору (Kubernetes) о проблеме до того, как на сервис будет подан трафик.30

### 4.1. Механизм ValidateOnStart

В ранних версиях.NET Core валидация опций происходила лениво (Lazy) — только при первом обращении к IOptions.Value. Это означало, что ошибка конфигурации могла всплыть спустя часы после деплоя, когда пользователь заходил на редко используемый эндпоинт.

В.NET 8 стандартной практикой является использование метода расширения ValidateOnStart(). Этот метод встраивается в цепочку регистрации опций и форсирует создание и валидацию объекта настроек на этапе построения DI-контейнера (Startup). Если валидация не проходит, выбрасывается исключение OptionsValidationException, и процесс приложения аварийно завершается, что является желаемым поведением для неработоспособного сервиса.31

### 4.2. Интеграция с FluentValidation и IValidateOptions

Хотя.NET поддерживает стандартные атрибуты Data Annotations (, , [Url]), они часто оказываются недостаточно гибкими для сложной бизнес-логики валидации (например, кросс-полевая валидация: "если SslEnabled равно true, то Port должен быть 443").

Профессиональным стандартом в Enterprise.NET является использование библиотеки **FluentValidation** в связке с паттерном Options. Для этого используется мост через интерфейс IValidateOptions<T>.

**Паттерн реализации валидации:**

1. **Определение Правил:** Создается класс валидатора, наследуемый от AbstractValidator<MySettings>, где декларативно описываются правила с использованием Fluent API (RuleFor(x => x.Key).NotEmpty()).33
2. **Адаптер:** Реализуется класс, имплементирующий интерфейс IValidateOptions<MySettings>. В методе Validate этого класса вызывается валидатор FluentValidation.
3. **Регистрация:** Адаптер регистрируется в DI-контейнере как Singleton.
4. **Активация:** При вызове ValidateOnStart(), ASP.NET Core автоматически находит все зарегистрированные реализации IValidateOptions для данного типа и выполняет проверку.16

Такой подход позволяет инкапсулировать логику валидации, отделив её от самих моделей данных (POCO), что соответствует принципу единственной ответственности (Single Responsibility Principle) и значительно упрощает Unit-тестирование правил валидации.16

## 5. Безопасность и Управление Секретами (Secret Management)

В разработке игр (Unity) разработчики часто допускают хранение ключей API, токенов и паролей прямо в коде (Hardcoding) или в сериализованных ассетах, полагаясь на бинарный формат сборки или обфускацию как на меру защиты. В серверной разработке (Enterprise) это категорически запрещено. Компрометация секретов (например, ключа от AWS/Azure или пароля от базы данных) может привести к полной потере данных и огромным финансовым убыткам.1

### 5.1. User Secrets (Среда Разработки)

Для локальной разработки (среда Development).NET предоставляет инструмент Secret Manager (команда CLI dotnet user-secrets).

Этот инструмент сохраняет секреты в JSON-файл (secrets.json), который физически расположен в системной директории профиля пользователя (%APPDATA%\Microsoft\UserSecrets на Windows или ~/.microsoft/usersecrets на Linux/macOS), а не в папке проекта.

В файл проекта (.csproj) добавляется только идентификатор UserSecretsId (GUID). При запуске приложения хост автоматически ищет этот файл по GUID и подгружает его в конфигурацию.

Гарантия безопасности: Даже если разработчик случайно выполнит git add. и git commit, реальные пароли не попадут в репозиторий, так как файла с секретами просто нет в рабочей директории.12

### 5.2. Паттерн KeyPerFile и Docker Secrets (Продакшен)

В продакшене, особенно при использовании оркестраторов контейнеров (Docker Swarm, Kubernetes), используется иной подход. Передача секретов через переменные окружения считается небезопасной, так как они могут быть случайно выведены в логи, доступны через docker inspect или видны в дампе процесса (/proc/{pid}/environ).17

Золотым стандартом безопасности является использование **файловых маунтов (File Mounts)**. Оркестратор монтирует секреты как файлы в специальную директорию в оперативной памяти (tmpfs/RAM-disk) внутри контейнера (обычно /run/secrets/ или /etc/secrets/).

* Имя файла соответствует имени ключа конфигурации.
* Содержимое файла — значению секрета.

Для интеграции этого механизма с.NET Configuration используется провайдер **KeyPerFile**.

C#

builder.Configuration.AddKeyPerFile(directoryPath: "/run/secrets", optional: true);

Этот провайдер сканирует указанную директорию и загружает каждый файл как пару ключ-значение в конфигурацию.

Нюанс с вложенностью (Nested Keys):

Файловая система в рамках одной директории плоская, а конфигурация.NET иерархична. Чтобы сопоставить файл секрета с вложенным свойством (например, RabbitMq:Password), имя файла должно использовать разделитель двойного подчеркивания: RabbitMq\_\_Password. Провайдер KeyPerFile (как и провайдер переменных окружения) автоматически транслирует \_\_ в : при чтении, корректно заполняя структуру классов Options.35 Это полная реализация паттерна envdir, обеспечивающая максимальный уровень изоляции секретов.11

## 6. Современные Тренды:.NET 8, Source Generators и Native AOT

### 6.1. Оптимизация через Source Generators

До версии.NET 8 методы привязки конфигурации (Configuration.Get<T>(), Bind()) опирались на механизм рефлексии (Reflection) для анализа свойств классов и заполнения их данными во время выполнения. Это создавало накладные расходы на производительность при старте приложения (Startup Time) и потребление памяти.

В.NET 8 был представлен генератор исходного кода для привязки конфигурации (Configuration Binding Source Generator). Этот механизм анализирует использование методов конфигурации (таких как Configure<T>, Bind, Get<T>) на этапе компиляции и генерирует оптимизированный C# код, который выполняет присваивание значений напрямую, без использования рефлексии.

Это приводит к двум важным следствиям:

1. **Ускорение запуска:** Исключение рефлексии делает старт приложения быстрее.
2. **Совместимость с Native AOT:** Native AOT (Ahead-of-Time compilation) компилирует приложение в нативный код, не требующий JIT-компилятора, но имеет ограничения на использование динамической рефлексии. Генераторы кода устраняют это ограничение, позволяя создавать сверхлегкие и быстрые контейнеры, идеально подходящие для Serverless (AWS Lambda, Azure Functions).38

Для студента это означает необходимость привыкать к использованию современных API и пониманию того, что магия связывания данных теперь происходит во время компиляции, а не во время выполнения.

### 6.2. Влияние на DevOps Пайплайны

Внедрение AOT и оптимизированных контейнеров меняет подход к CI/CD. Сборка AOT-приложения занимает больше времени и требует больше ресурсов (CPU/RAM) на билд-агенте, чем обычная сборка.NET, но дает выигрыш в рантайме. Это требует более тщательной настройки кэширования слоев Docker и этапов восстановления пакетов (NuGet Restore), чтобы не замедлять цикл обратной связи (Feedback Loop) разработчика.25

## 7. Практическая Реализация: От Теории к Коду

Понимание теоретических основ первого дня обучения закладывает фундамент для всех последующих активностей недели. Разработчик должен не просто "заставить код работать", но спроектировать систему, устойчивую к изменениям среды, безопасную по дизайну и эффективную по ресурсам.

### Чек-лист для самопроверки архитектурных решений:

1. **Environment Agnostic:** Есть ли в коде проверки if (env.IsProduction()) для логики конфигурации? (Должно быть минимизировано, конфигурация должна инжектиться снаружи).
2. **Secret Isolation:** Есть ли в appsettings.json реальные пароли? (Должны быть только заглушки или пустые строки).
3. **Fail Fast:** Упадет ли сервис сразу, если я удалю секцию настройки базы данных? (Должен упасть с внятной ошибкой валидации).
4. **Allocation Awareness:** Использую ли я IOptionsSnapshot в высоконагруженном Middleware? (Стоит рассмотреть IOptionsMonitor или IOptions для Singleton-компонентов).

Этот день знаменует собой переход от мышления "разработчика клиента" к мышлению "инженера распределенных систем", где конфигурация — это контракт между кодом и инфраструктурой.

#### Источники

1. План обучения CI/CD и конфигурации .NET неделя 13
2. Separate Game Data and Logic with ScriptableObjects - Unity, дата последнего обращения: декабря 6, 2025, <https://unity.com/how-to/separate-game-data-logic-scriptable-objects>
3. 6 ways ScriptableObjects can benefit your team and your code - Unity, дата последнего обращения: декабря 6, 2025, <https://unity.com/blog/engine-platform/6-ways-scriptableobjects-can-benefit-your-team-and-your-code>
4. Confusion regarding scriptableObject - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/58798846/confusion-regarding-scriptableobject>
5. How do environment variables and appSettings file(s) get used during publish?, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/55010022/how-do-environment-variables-and-appsettings-files-get-used-during-publish>
6. CI/CD with Jenkins pipelines, part 1: .NET Core application deployments on AWS ECS, дата последнего обращения: декабря 6, 2025, <https://devblog.xero.com/ci-cd-with-jenkins-pipelines-part-1-net-core-application-deployments-on-aws-ecs-987b8e032aa0>
7. Configuration - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration>
8. Options pattern in ASP.NET Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-10.0>
9. Scriptable Objects vs GameObject MonoBehaviours - Unity Tutorial for Beginners 2023, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=XofEVlNmWlI>
10. .NET Core Configuration in .NET 8 | by Dhananjeyan Balaretnaraja | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@sdbala/net-core-configuration-in-net-8-4a8365f24ff1>
11. Change ASP.NET Core application configuration sources order of precedence, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/79184796/change-asp-net-core-application-configuration-sources-order-of-precedence>
12. Order of Precedence when Configuring ASP.NET Core - Microsoft Developer Blogs, дата последнего обращения: декабря 6, 2025, <https://devblogs.microsoft.com/premier-developer/order-of-precedence-when-configuring-asp-net-core/>
13. Configuration in ASP.NET Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-10.0>
14. ASP.NET 5 - A Deep Dive into the ASP.NET 5 Runtime | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/archive/msdn-magazine/2015/march/asp-net-5-a-deep-dive-into-the-asp-net-5-runtime>
15. A Beginner's Guide to .NET's HostBuilder: Part 4 — Configurations and Environments | by Sawyer Watts | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@sawyer.watts/a-beginners-guide-to-net-s-hostbuilder-part-3-configurations-and-environments-61a8b7c1867a>
16. Options Pattern Validation in ASP.NET Core With FluentValidation, дата последнего обращения: декабря 6, 2025, <https://www.milanjovanovic.tech/blog/options-pattern-validation-in-aspnetcore-with-fluentvalidation>
17. Safe storage of app secrets in development in ASP.NET Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-10.0>
18. Override array settings in appsettings.json with those in appsettings.Production.json - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/52755027/override-array-settings-in-appsettings-json-with-those-in-appsettings-production>
19. Unexpected behavior at .Net Core AppSettings.json file reader - Developer Community, дата последнего обращения: декабря 6, 2025, <https://developercommunity.visualstudio.com/content/problem/579268/unexpected-behavior-at-net-core-appsettingsjson-fi.html>
20. Default configuration array merging is confusing and error-prone · Issue #118204 · dotnet/runtime - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/dotnet/runtime/issues/118204>
21. How to merge multiple arrays from ASP.Net Core configuration files? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/53985133/how-to-merge-multiple-arrays-from-asp-net-core-configuration-files>
22. The Hidden Rule of Configuration Keys in .NET Core and Azure Portal - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@phamduckhanh2411/the-hidden-rule-of-configuration-keys-in-net-core-and-azure-portal-9410f9155a6d>
23. Understanding IOptions, IOptionsSnapshot, and IOptionsMonitor - Felipe Gavilán, дата последнего обращения: декабря 6, 2025, <https://gavilan.blog/2025/03/25/understanding-ioptions-ioptionssnapshot-and-ioptionsmonitor/>
24. IOptions vs IOptionsSnapshot vs IOptionsMonitor in .NET — What's the Difference?, дата последнего обращения: декабря 6, 2025, <https://medium.com/c-sharp-programming/ioptions-vs-ioptionssnapshot-vs-ioptionsmonitor-dotnet-6f2305b09770>
25. Difference between IOptionsMonitor vs. IOptionsSnapshot - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/50788988/difference-between-ioptionsmonitor-vs-ioptionssnapshot>
26. IOptionsSnapshot is very slow · Issue #53793 · dotnet/runtime - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/dotnet/runtime/issues/53793>
27. Detect changes with change tokens in ASP.NET Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/change-tokens?view=aspnetcore-10.0>
28. Usage of IOptionsSnapshot is slow (replace with IOptionsMonitor?) · Issue #10506 - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/abpframework/abp/issues/10506>
29. Difference between events and change tokens in C# - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/68833769/difference-between-events-and-change-tokens-in-c-sharp>
30. Integrating FluentValidation with the Options Pattern in ASP.NET Core - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@serasiyasavan14/integrating-fluentvalidation-with-the-options-pattern-in-asp-net-core-2a674b4d0890>
31. Validation of ASP.NET Core options during startup - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/51692665/validation-of-asp-net-core-options-during-startup>
32. ValidateOnStart broken in ASP.NET 8? : r/csharp - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/csharp/comments/1dob4bo/validateonstart_broken_in_aspnet_8/>
33. FluentValidation — FluentValidation documentation, дата последнего обращения: декабря 6, 2025, <https://fluentvalidation.net/>
34. Options pattern - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/extensions/options>
35. Configuration providers - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration-providers>
36. KeyPerFileConfigurationBuilderE, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.keyperfileconfigurationbuilderextensions.addkeyperfile?view=net-10.0-pp>
37. Using Key-per-file configuration provider - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/karenpayneoregon/using-key-per-file-configuration-provider-a9h>
38. Compile-time configuration source generation - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration-generator>
39. Microsoft.Extensions.Configuration.Binder 8.0.1 - NuGet, дата последнего обращения: декабря 6, 2025, <https://www.nuget.org/packages/Microsoft.Extensions.Configuration.Binder/8.0.1>
40. What's new in .NET 8 runtime - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8/runtime>
41. Performance Improvements in .NET 8 | by Rico Mariani - Medium, дата последнего обращения: декабря 6, 2025, <https://ricomariani.medium.com/performance-improvements-in-net-8-f673e805e09e>