# Архитектурный Анализ Управления Секретами и Паттерна envdir в Облачно-Нативных Приложениях.NET Core

## Введение: Эволюция Парадигмы Конфигурации

Переход от разработки клиентских приложений на Unity к инженерным практикам построения серверных систем уровня Enterprise на платформе.NET Core, как это обозначено в учебном плане 13-й и 18-й недель, требует фундаментальной деконструкции ментальных моделей, касающихся жизненного цикла приложения и управления его состоянием. В центре внимания "Второго дня" учебного модуля находится одна из самых критических и часто недооцененных дисциплин в современной разработке программного обеспечения — безопасное и архитектурно корректное управление секретами и конфигурацией.1

В экосистеме GameDev, в частности при работе с Unity, конфигурация традиционно рассматривается как статический актив. Параметры геймплея, настройки физики или даже адреса серверов часто "запекаются" (baked) непосредственно в бинарные файлы игры через ScriptableObjects, префабы или сериализованные поля сцен.1 Такой подход оправдан для монолитных клиентских приложений, распространяемых через App Store или Steam, где обновление конфигурации эквивалентно выпуску патча. Однако при переносе этой ментальности в мир Backend-разработки возникают катастрофические архитектурные уязвимости. Серверное приложение должно быть stateless (без сохранения состояния) и environment-agnostic (независимым от среды исполнения). Один и тот же Docker-образ должен корректно функционировать как на локальной машине разработчика, так и в тестовом контуре (QA), и в высоконагруженном продакшене, изменяя свое поведение исключительно за счет внешней инъекции конфигурации.1

Настоящий отчет представляет собой исчерпывающий анализ теоретических основ и практических механизмов реализации безопасного управления конфигурацией. Мы подробно рассмотрим методологию 12-Factor App, проанализируем уязвимости традиционных переменных окружения, обоснуем переход к файловому паттерну envdir и детально разберем интеграцию этих подходов в экосистему ASP.NET Core с использованием провайдеров KeyPerFileConfigurationProvider и User Secrets.

## Глава 1. Теоретический Базис: Конфигурация как Внешняя Зависимость

### 1.1. Принцип Третьего Фактора (Store Config in the Environment)

Основополагающим документом для построения современных облачных приложений является манифест "The Twelve-Factor App". Третий фактор этой методологии гласит: "Сохраняйте конфигурацию в среде выполнения".3 Для Unity-разработчика, привыкшего к визуальной настройке параметров в инспекторе редактора, это требование может показаться контритуитивным, однако оно является единственным способом обеспечить масштабируемость и безопасность.

Под "конфигурацией" в данном контексте понимается все, что может варьироваться между развертываниями (деплоями). Сюда входят:

* **Дескрипторы ресурсов:** Строки подключения к базам данных, хосты брокеров сообщений (RabbitMQ), адреса кэширующих серверов (Redis).4
* **Учетные данные:** Пароли, API-токены, секретные ключи шифрования (Symmetric Security Keys), SSH-ключи.4
* **Канонические параметры среды:** Внешние хостнеймы, настройки CORS, флаги функциональности (Feature Flags).1

Критически важно отличать конфигурацию от внутренней настройки приложения. Например, маппинг маршрутов в контроллерах или связывание модулей в DI-контейнере не является конфигурацией в понимании 12-Factor, так как эти связи неизменны при переходе от Staging к Production.4

Лакмусовой бумажкой для проверки правильности архитектуры конфигурации является следующий мысленный эксперимент: "Можете ли вы в любой момент открыть исходный код вашего приложения в публичном репозитории (Open Source), не скомпрометировав при этом никаких учетных данных?".4 Если ответ отрицательный, и в коде присутствуют хардкодированные пароли или токены, приложение нарушает принципы безопасности и требует немедленного рефакторинга.

### 1.2. Иерархия Источников Конфигурации в.NET

Платформа.NET Core (и.NET 5/6/7/8) реализует гибкую, слоистую архитектуру конфигурации, которая идеально ложится на требования 12-Factor App. При инициализации хоста через WebApplication.CreateBuilder(args) 5 создается ConfigurationManager, который агрегирует настройки из множества источников. Понимание порядка приоритетов этих источников критически важно для Backend-инженера.

Источники загружаются в следующем порядке (от низшего приоритета к высшему):

1. **appsettings.json**: Базовая конфигурация, содержащая безопасные значения по умолчанию.1
2. **appsettings.{Environment}.json**: Настройки, специфичные для текущей среды (например, appsettings.Development.json). Эти значения переопределяют базовые.1
3. **User Secrets (Секреты пользователя)**: Применяются только в среде Development. Позволяют разработчику переопределять настройки локально, не затрагивая файлы проекта.5
4. **Переменные окружения (Environment Variables)**: Значения, установленные операционной системой или контейнером. Они переопределяют все файловые настройки.5
5. **Аргументы командной строки (CLI Arguments)**: Имеют наивысший приоритет, позволяя переопределять любые параметры при запуске процесса.6

Однако в контексте "Второго дня" обучения мы вводим новый, специализированный источник, который интегрируется в эту цепочку для работы в контейнеризированных средах — **Key-Per-File Configuration Provider** (Провайдер конфигурации "Ключ-в-Файле"). Его место в иерархии зависит от момента регистрации, но архитектурно он призван заменить или дополнить переменные окружения, обеспечивая более высокий уровень безопасности.6

## Глава 2. Угрозы Безопасности и Недостатки Переменных Окружения

Хотя переменные окружения являются стандартом де-факто для передачи конфигурации, при работе с чувствительными данными (секретами) они обладают рядом существенных недостатков, которые делают их использование в чистом виде рискованным в Enterprise-системах.

### 2.1. Вектор Атаки через /proc/self/environ

В операционных системах Linux (на базе которых работает большинство Docker-контейнеров) каждая переменная окружения процесса доступна через виртуальную файловую систему /proc. Файл /proc/<PID>/environ содержит полное окружение процесса с идентификатором PID.9

Это создает серьезный вектор атаки. Если злоумышленник получает возможность чтения файлов на сервере (например, через уязвимость Path Traversal или LFI — Local File Inclusion) или получает доступ к мониторингу процессов, он может тривиально выгрузить все секреты приложения, просто прочитав этот файл. В отличие от файлов конфигурации, права доступа к которым можно строго ограничить, доступ к /proc регулируется ядром и часто доступен другим процессам того же пользователя.9

### 2.2. Утечки в Логах и Дочерних Процессах

Второй критической проблемой является наследование окружения. Когда приложение порождает дочерний процесс (например, вызывает внешний скрипт, утилиту миграции БД или отправляет отчет об ошибке), дочерний процесс автоматически наследует все переменные окружения родителя.11 Это означает, что сторонний инструмент, которому не нужны права доступа к базе данных, все равно получает строку подключения в своем окружении. Если этот инструмент скомпрометирован или просто некорректно логирует свое состояние при ошибке, секреты могут утечь в журналы (logs).11

Системы агрегации логов (ELK Stack, Splunk, Datadog) часто сохраняют данные в открытом виде. Попадание пароля от продакшен-базы в логи делает его доступным для широкого круга разработчиков и админов, что нарушает принцип наименьших привилегий.11

### 2.3. Видимость в Docker Inspect

При запуске контейнера с передачей переменных через флаг -e или инструкцию ENV в Dockerfile, эти значения навсегда сохраняются в метаданных контейнера. Команда docker inspect <container\_id> выдаст JSON, содержащий все переменные окружения в открытом виде.13 Любой пользователь, имеющий доступ к демону Docker (даже в режиме read-only), сможет прочитать секреты. Это делает использование простых переменных окружения неприемлемым для передачи паролей и ключей шифрования в высокозащищенных средах.

## Глава 3. Паттерн envdir: Архитектура Файловой Инъекции

Ответом на уязвимости переменных окружения стал паттерн envdir, исторически восходящий к набору инструментов daemontools, разработанному Daniel J. Bernstein.14 Этот паттерн предлагает использовать файловую систему как примитив для хранения конфигурации.

### 3.1. Философия "Все есть файл"

В концепции envdir директория представляет собой среду (environment). Каждый файл внутри этой директории соответствует одной переменной конфигурации:

* **Имя файла** — это ключ (имя переменной).
* **Содержимое файла** — это значение переменной.14

Например, структура директории может выглядеть так:

/etc/secrets/

├── DATABASE\_PASSWORD (содержит: super\_secret\_pass)

├── API\_KEY (содержит: xy78234kb)

└── RABBITMQ\_HOST (содержит: rabbitmq.prod.local)

16

Утилита envdir считывает эту директорию, загружает содержимое файлов в память процесса и запускает приложение. Однако в современных контейнерных оркестраторах (Docker Swarm, Kubernetes) этот паттерн реализован нативно через механизм **Secrets**.

### 3.2. Docker Secrets и Монтирование tmpfs

Docker Secrets реализует паттерн envdir, используя возможности ядра Linux по созданию временных файловых систем в оперативной памяти (tmpfs/ramfs).

Когда секрет назначается сервису в Docker Swarm или через Docker Compose, он не записывается на диск контейнера. Вместо этого он монтируется в директорию /run/secrets/.18 Эта директория находится исключительно в оперативной памяти (RAM).

Это дает колоссальные преимущества в безопасности:

1. **Эфемерность**: При остановке контейнера секреты исчезают. Они не остаются на диске, что исключает их восстановление при криминалистическом анализе выключенного сервера.13
2. **Изоляция**: Секреты не видны через docker inspect и не попадают в переменные окружения, что защищает от утечек через /proc и наследование процессов.13
3. **Гранулярный контроль доступа**: Доступ к файлам в /run/secrets регулируется стандартными правами доступа Linux (permissions), что позволяет ограничить чтение только для владельца процесса.18

## Глава 4. Интеграция с.NET: KeyPerFileConfigurationProvider

Для того чтобы приложение на.NET Core могло прозрачно потреблять секреты, организованные по паттерну envdir (Docker Secrets), платформа предоставляет специализированный провайдер конфигурации — KeyPerFileConfigurationProvider.6

### 4.1. Механизм Работы AddKeyPerFile

В Program.cs разработчик подключает этот провайдер с помощью метода расширения builder.Configuration.AddKeyPerFile. Этот метод принимает путь к директории и сканирует её содержимое, превращая файлы в пары "Ключ-Значение" в объекте IConfiguration.6

Пример реализации, адаптированный под учебный план "Второго дня" 1:

C#

var secretsPath = "/run/secrets";  
// Проверка существования директории критична для кросс-платформенной совместимости (Dev vs Prod)  
if (Directory.Exists(secretsPath))  
{  
 builder.Configuration.AddKeyPerFile(directoryPath: secretsPath, optional: true, reloadOnChange: true);  
}

1

Параметр optional: true позволяет приложению не падать, если директория пуста или отсутствует (например, при локальном запуске без Docker), а reloadOnChange: true активирует подписку на изменения файлов, что позволяет обновлять конфигурацию "на лету" без перезапуска приложения.6

### 4.2. Проблема Иерархии и Двойное Подчеркивание (\_\_)

Одной из главных сложностей интеграции Linux-контейнеров и.NET является различие в синтаксисе иерархии.

В конфигурации.NET (appsettings.json) иерархия обозначается двоеточием (:), например: RabbitMq:Password.

Однако в файловых системах Linux и именах переменных окружения двоеточие часто является недопустимым символом или разделителем путей, что делает создание файла с именем RabbitMq:Password проблематичным.5

Для решения этой проблемы используется конвенция **Двойного Подчеркивания (\_\_)**. Провайдеры конфигурации.NET (как EnvironmentVariables, так и KeyPerFile) автоматически транслируют \_\_ в : при чтении ключей.6

Следовательно, для того чтобы передать значение в секцию RabbitMq свойство Password, файл секрета в Docker должен называться RabbitMq\_\_Password.

* **Имя файла:** /run/secrets/RabbitMq\_\_Password
* **Содержимое:** super\_secure\_pass
* **Результат в.NET:** Configuration = "super\_secure\_pass"

Это позволяет корректно маппить плоскую структуру файлов на глубоко вложенные объекты опций (Options Pattern), такие как RabbitMqOptions или DatabaseOptions.22

### 4.3. Динамическая Перезагрузка (Hot Reload)

Особенностью использования KeyPerFile с параметром reloadOnChange: true является возможность ротации секретов без простоя сервиса (Zero Downtime). Если оркестратор (например, Kubernetes) обновит содержимое файла секрета (по сути, пересоздаст симлинк на новую версию секрета), FileProvider в.NET обнаружит изменение.

В сочетании с использованием интерфейса IOptionsSnapshot<T> (который пересоздает объект настроек на каждый запрос) или IOptionsMonitor<T> (который предоставляет событие OnChange), приложение мгновенно начнет использовать новый пароль для новых подключений.1 Это фундаментальное преимущество перед переменными окружения, изменение которых требует перезапуска процесса.

## Глава 5. Управление Секретами в Среде Разработки (User Secrets)

Поскольку монтирование /run/secrets является спецификой контейнерной среды, возникает вопрос: как обеспечить безопасное хранение секретов на машине разработчика, где Docker может не использоваться для запуска (например, при отладке через dotnet run или Visual Studio)? Здесь на сцену выходят **User Secrets** (Секреты пользователя).1

### 5.1. Архитектура и Хранение

User Secrets — это инструмент, который позволяет хранить конфигурационные файлы вне дерева проекта. Это гарантирует, что разработчик случайно не закоммитит файл appsettings.json с реальными паролями в Git.

При инициализации секретов командой dotnet user-secrets init 8, в файл проекта (.csproj) добавляется свойство UserSecretsId:

XML

<PropertyGroup>  
 <UserSecretsId>79a3edd0-2092-40a2-a04d-dcb46d5ca9ed</UserSecretsId>  
</PropertyGroup>

8

Этот GUID служит идентификатором папки в профиле пользователя ОС, где физически хранится JSON-файл с секретами (secrets.json).

**Локации хранения:**

* **Windows:** %APPDATA%\Microsoft\UserSecrets\<UserSecretsId>\secrets.json.8
* **Linux / macOS:** ~/.microsoft/usersecrets/<UserSecretsId>/secrets.json.26

### 5.2. Ограничения Безопасности

Критически важно понимать, что User Secrets **не шифруются**. Это обычный текстовый JSON-файл.8 Механизм защищает только от утечки через систему контроля версий (Git), но не от вредоносного ПО на компьютере разработчика. Поэтому User Secrets предназначены исключительно для среды **Development**. Использование их в Production категорически запрещено.5

В коде WebApplication.CreateBuilder провайдер User Secrets добавляется автоматически, но только если переменная среды ASPNETCORE\_ENVIRONMENT установлена в значение Development.8 Это предотвращает случайное использование отладочных паролей на боевых серверах.

## Глава 6. Практическая Оркестрация: Docker Compose

Связующим звеном между теорией envdir и практикой.NET является файл docker-compose.yml. Именно здесь мы определяем, как абстрактные секреты проецируются в файлы внутри контейнера.

### 6.1. Синтаксис Объявления Секретов

В Docker Compose версии 3.x+ секреты определяются в двух секциях: глобальной (bottom-level) и на уровне сервиса.18

Пример конфигурации для сервиса "Calendar Service" 1:

YAML

version: '3.8'  
  
services:  
 calendar-service:  
 image: calendar-service:latest  
 environment:  
 - ASPNETCORE\_ENVIRONMENT=Production  
 secrets:  
 - source: rabbitmq\_pass # Ссылка на глобальный секрет  
 target: RabbitMq\_\_Password # Имя файла внутри контейнера (для KeyPerFile)  
  
secrets:  
 rabbitmq\_pass:  
 file:./.secrets\_mock/rabbitmq\_password.txt # Локальная симуляция файла

### 6.2. Роль Атрибута Target

Обратите внимание на атрибут target: RabbitMq\_\_Password. Это ключевой момент интеграции. Глобальный секрет может называться как угодно (например, rabbitmq\_pass), но внутри контейнера он должен появиться под именем, которое KeyPerFileConfigurationProvider сможет распознать и транслировать в иерархию.NET (то есть с двойным подчеркиванием).22 Без этого явного переименования через target, файл будет называться rabbitmq\_pass, и конфигурация.NET воспримет его как корневой ключ, что сломает привязку к классу RabbitMqOptions.

### 6.3. Симуляция Среды (Mocking)

Для локальной разработки, максимально приближенной к продакшену (Dev/Prod Parity), используется техника "мокинга" секретов. Создается локальная директория .secrets\_mock, содержащая текстовые файлы с паролями. Эти файлы добавляются в .gitignore. В docker-compose.yml секреты объявляются через file:./.secrets\_mock/....

Это позволяет разработчику тестировать механизм монтирования секретов и работу KeyPerFileProvider локально, не разворачивая полноценный кластер Docker Swarm.1

## Глава 7. Продвинутые Аспекты и Граничные Случаи

### 7.1. Конфликт Прав Доступа (Root vs App User)

Одной из самых частых проблем при переходе на безопасные образы.NET 8 (которые запускаются от имени пользователя app, а не root) является ошибка UnauthorizedAccessException при попытке чтения секретов.32

По умолчанию Docker монтирует секреты в /run/secrets с владельцем root и правами, часто ограниченными до 0444 или даже 0400 (только чтение для владельца). Если приложение работает под пользователем app (UID 1654), оно не может прочитать эти файлы.

**Стратегии решения:**

1. **Настройка UID/GID в Compose:** Явное указание, что секрет должен принадлежать пользователю app.
2. **Файловые права (File Mode):** Установка прав 0444 (чтение для всех), что допустимо внутри изолированного контейнера.18
3. **Запуск от Root с понижением прав:** (Менее безопасно) Запуск контейнера от root, чтение секретов, и затем программное понижение привилегий, но это противоречит принципам использования distroless/chiseled образов.

### 7.2. Приоритет Перезаписи (Precedence)

Важно помнить, что KeyPerFileConfigurationProvider должен быть добавлен в ConfigurationBuilder **после** провайдеров JSON и переменных окружения, чтобы гарантировать, что секреты из защищенного хранилища переписывают любые значения по умолчанию или случайно забытые переменные среды.5 В стандартном WebApplication.CreateBuilder порядок жестко задан, поэтому для корректного переопределения часто приходится манипулировать коллекцией Sources или добавлять провайдер явно в конце цепочки инициализации.6

## Таблицы Сравнительного Анализа

### Таблица 1. Сравнение Механизмов Передачи Конфигурации

| **Характеристика** | **appsettings.json** | **Environment Variables** | **User Secrets** | **Docker Secrets (envdir)** |
| --- | --- | --- | --- | --- |
| **Основной сценарий** | Дефолтные настройки, структура | Системные параметры, флаги | Локальная разработка (Dev) | Продакшен (Sensitive Data) |
| **Место хранения** | Диск (в репозитории Git) | Блок окружения процесса (RAM) | Папка профиля пользователя (Disk) | Виртуальная ФС (tmpfs/RAM) |
| **Безопасность** | **Низкая** (Риск коммита в Git) | **Средняя** (Видны в /proc, logs, inspect) | **Высокая** (для Dev), **Низкая** (для Prod) | **Максимальная** (Шифрование at rest, tmpfs) |
| **Формат данных** | Иерархический JSON | Плоские строки (разделитель \_\_) | Иерархический JSON | Файлы (имя=ключ, контент=значение) |
| **Hot Reload** | Поддерживается | Нет (Требует рестарта) | Поддерживается | Поддерживается (через IOptionsSnapshot) |

### Таблица 2. Реализация 12-Factor Config в.NET (Чек-лист)

| **Принцип** | **Реализация в.NET Core** | **Инструмент** |
| --- | --- | --- |
| **Строгое разделение конфига и кода** | Исключение appsettings.json из Git (через .gitignore) или использование шаблонов | .gitignore, appsettings.Template.json |
| **Конфигурация в среде выполнения** | Использование переменных окружения и файловых маунтов | EnvironmentVariablesProvider, KeyPerFileProvider |
| **Паритет Dev/Prod** | Использование абстракции IConfiguration, скрывающей источник | ConfigurationBuilder, User Secrets (Dev) -> Docker Secrets (Prod) |
| **Безопасность секретов** | Изоляция секретов от файловой системы контейнера | Docker Secrets (tmpfs mount), KeyPerFile |

## Заключение

Анализ теоретической базы "Второго дня" демонстрирует, что управление секретами в Enterprise-системах — это многослойная архитектурная задача, не имеющая ничего общего с простым "сохранением настроек". Переход от Unity-паттернов к облачно-нативным практикам требует внедрения строгой дисциплины: отказ от хардкода, использование локальных хранилищ секретов (User Secrets) при разработке и применение файловых инъекций (Docker Secrets / envdir) в продакшене.

Интеграция KeyPerFileConfigurationProvider в.NET обеспечивает элегантный мост между инфраструктурными требованиями безопасности (хранение секретов в оперативной памяти) и удобством разработки (прозрачный доступ через типизированные опции). Понимание этих механизмов является обязательным условием для создания надежных, защищенных и масштабируемых Backend-сервисов, соответствующих современным индустриальным стандартам.

#### Источники

1. План обучения CI/CD и конфигурации .NET неделя 13
2. V. Build, release, run - The Twelve-Factor App, дата последнего обращения: декабря 6, 2025, <https://12factor.net/build-release-run>
3. The Twelve-Factor App, дата последнего обращения: декабря 6, 2025, <https://12factor.net/>
4. Store config in the environment - The Twelve-Factor App, дата последнего обращения: декабря 6, 2025, <https://12factor.net/config>
5. 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>
6. Configuration providers - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration-providers>
7. configuration | chrisxfires blog, дата последнего обращения: декабря 6, 2025, <https://chriscodes.net/notes/asp.net/fundamentals/configuration/>
8. 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>
9. Understanding security implications of secrets vs. env vars in Docker Compose, дата последнего обращения: декабря 6, 2025, <https://forums.docker.com/t/understanding-security-implications-of-secrets-vs-env-vars-in-docker-compose/145903>
10. Why is Docker Secrets more secure than environment variables? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/44615837/why-is-docker-secrets-more-secure-than-environment-variables>
11. Do not use secrets in environment variables and here's how to do it better, дата последнего обращения: декабря 6, 2025, <https://www.nodejs-security.com/blog/do-not-use-secrets-in-environment-variables-and-here-is-how-to-do-it-better>
12. Is it secure to store passwords as environment variables (rather than as plain text) in config files? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/12461484/is-it-secure-to-store-passwords-as-environment-variables-rather-than-as-plain-t>
13. Docker security risks passing secrets over environment variables - DevOps Stack Exchange, дата последнего обращения: декабря 6, 2025, <https://devops.stackexchange.com/questions/5077/docker-security-risks-passing-secrets-over-environment-variables>
14. envdir(8) — daemontools — Debian testing, дата последнего обращения: декабря 6, 2025, <https://manpages.debian.org/testing/daemontools/envdir.8.en.html>
15. Celebrating Daemontools - Infinite Negative Utility, дата последнего обращения: декабря 6, 2025, <https://journal.infinitenegativeutility.com/celebrating-daemontools>
16. envdir Documentation, дата последнего обращения: декабря 6, 2025, <https://envdir.readthedocs.io/_/downloads/en/latest/pdf/>
17. envdir (Python port) — envdir 0.7 documentation, дата последнего обращения: декабря 6, 2025, <https://envdir.readthedocs.io/>
18. Secrets in Compose - Docker Docs, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/compose/how-tos/use-secrets/>
19. 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>
20. 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>
21. KeyPerFileConfigurationSource Class (Microsoft.Extensions.Configuration.KeyPerFile), дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.keyperfile.keyperfileconfigurationsource?view=net-10.0-pp>
22. Key-per-file config in .NET Core - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/68846952/key-per-file-config-in-net-core>
23. How does reloadOnChange of Microsoft.Extensions.Configuration work for appsettings.json, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/43341058/how-does-reloadonchange-of-microsoft-extensions-configuration-work-for-appsettin>
24. How to get "Manage User Secrets" in a .NET Core console-application? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/42268265/how-to-get-manage-user-secrets-in-a-net-core-console-application>
25. Storing application secrets safely during development - .NET | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/secure-net-microservices-web-applications/developer-app-secrets-storage>
26. Everything you need to know about configuration and secret management in .NET, дата последнего обращения: декабря 6, 2025, <https://stenbrinke.nl/blog/configuration-and-secret-management-in-dotnet/>
27. Moving .NET Core user secrets to another computer - Joonas Westlin, дата последнего обращения: декабря 6, 2025, <https://joonasw.net/view/moving-net-core-user-secrets-to-another-computer>
28. What is the purpose of the UserSecretsId in the .csproj file, and should it be hidden or public? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/78760116/what-is-the-purpose-of-the-usersecretsid-in-the-csproj-file-and-should-it-be-h>
29. User Secrets not loading with WebApplication.CreateBuilder(args) - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/75479681/user-secrets-not-loading-with-webapplication-createbuilderargs>
30. Build secrets - Docker Docs, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/build/building/secrets/>
31. Manage sensitive data with Docker secrets, дата последнего обращения: декабря 6, 2025, <https://docs.docker.com/engine/swarm/secrets/>
32. Unable to Access Secrets Files from Container - docker - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/docker/comments/m9afto/unable_to_access_secrets_files_from_container/>
33. Unable set docker secret file ownership, permissions - General, дата последнего обращения: декабря 6, 2025, <https://forums.docker.com/t/unable-set-docker-secret-file-ownership-permissions/31336>