Информационные процессы, Том 23, № 3, 2023, стр. 379–384

ИНФОРМАЦИОННОЕ ВЗАИМОДЕЙСТВИЕ

О некоторых особенностях переноса Windows web приложений в среду Unix[1]

К.П. Погорелко

Межведомственный суперкомпьютерный центр РАН
konstpog@yandex.ru

Поступила в редколлегию 28.08.2023

Аннотация—В статье рассматриваются вопросы переноса web приложения, реализованного в кроссплатформенной системе .NET Core из операционной системы Windows в операционную систему Unix. Рассмотрены аспекты развертывания приложения, миграции на другой SQL сервер и проблемы, возникающие с библиотеками обработки изображений. Делается вывод о том, что использование кроссплатформенных систем для разработки приложений позволяет достаточно эффективно разрабатывать приложения, функционирующие на разных серверных платформах, однако перенос системы на другую серверную платформу может вызвать определенные проблемы.

КЛЮЧЕВЫЕ СЛОВА: информационные системы, web приложения, .NET Core, кроссплатформенные системы, проблемы переноса, обработка изображений

DOI: 10.53921/18195822_2023_23_3_379

1. ВВЕДЕНИЕ

В настоящее время стали широко применяться кроссплатформенные системы программирования, т.е. такие системы, которые позволяют создавать программы способные работать на разных операционных системах. Использование таких систем дает ряд преимуществ при разработке программных продуктов.

Во-первых, в процессе развития информационных систем возникают потребности перевода их программных оболочек с одной операционной платформы на другую. Это касается, в частности, актуальной задачи перехода с операционных систем семейства Windows на операционные системы семейства UNIX. Во-вторых, работа в кроссплатформенной системе позволяет программисту оставаться в рамках привычной среды, независимо от того, для какой платформы разрабатывается программный продукт, что повышает производительность и упрощает дальнейшее сопровождение этого продукта.

Однако перевод работающей системы на другую операционную платформу ставит такие проблемы, как обеспечение среды выполнения, обеспечение доступа к системе, адаптация системы манипулирования данными. Особенно это актуально для систем, оперирующих как с метаданными объектов, так и с изображениями и мультимедиа.

Задачу перевода действующей крупной информационной системы на новую операционную платформу рассмотрим на примере системы “Электронная библиотека Научное наследие России” ЭБ ННР [1].

[1] Работа выполняется в МСЦ РАН в рамках государственного задания по теме FNEF-2023-0014.

2. ПОСТАНОВКА ЗАДАЧИ

ЭБ ННР первоначально была реализована в 2009 г. [2, 3] как ряд взаимодействующих между собой модулей, функционировавших на различных платформах [4]. Так, подготовка метаданных осуществлялась в системе SciRus [5] под управлением Unix, презентация метаданных осуществлялась с использованием программной оболочки Единого Научного Информационного Пространства РАН (ЕНИП РАН) [6], подготовка и презентация полнотекстовых документов осуществлялась системой, реализованной на платформе .NET под Windows [7]. Этот подход был оправдан, поскольку была поставлена задача запустить систему в кратчайшие сроки, и для формирования ее программной оболочки использовались существовавшие к тому времени решения, разработанные в ВЦ РАН и БЕН РАН. В течение почти 10 лет система успешно функционировала, но наступило время, когда такая архитектура стала препятствием для ее дальнейшего развития.

В 2019 г. было принято решение о переводе программного обеспечения системы на единую современную платформу.

3. ВЫБОР РЕШЕНИЯ

Реализация системы была начата в кроссплатформенной среде Microsoft .NET Core 3.1 под операционной системой Windows [8], поскольку разработчик (он же автор одного из блоков первой версии системы) имел большой опыт работы в этой среде.

Платформа .NET Core – это открытая универсальная платформа разработки, которая поддерживается корпорацией Майкрософт и сообществом .NET на сайте GitHub. Эта система позволяет разрабатывать приложения, которые могут функционировать под операционными системами Windows, Mac OS и Linux, а также может использоваться как на локальных устройствах, так и в облаке на серверных платформах. Особенностью платформы .NET Core является то, что подготовленный к развертыванию проект компилируется в виде единого исполняемого файла. Это решение является эффективным с точки зрения производительности, т.к. полностью исключается процесс интерпретации в процессе выполнения, но любое исправление требует перекомпиляции проекта и его повторной публикации [9].

Разработка проекта велась на Microsoft Visual Studio. Visual Studio – это комплексная интегрированная среда разработки (IDE), которую можно использовать для написания, редактирования, отладки и сборки кода, а затем для развертывания приложения. Помимо редактирования и отладки кода, Visual Studio включает компиляторы, средства завершения кода, систему управления версиями, расширения и многие другие функции для улучшения каждого этапа процесса разработки программного обеспечения [10].

На момент развертывания системы было поставлено условие работы системы под операционной системой Unix. Для развертывания проекта была выбрана версия Unix – Ubuntu. Для запуска скомпилированного .NET Core проекта на сервере Ubuntu необходимо установить пакет 'dotnet' версии, соответствующей использованной версии .NET Core. Этот пакет обеспечивает исполнение скомпилированного проекта .NET на соответствующей операционной системе и доступен практически во всех Unix библиотеках .

При развертывании web-проекта необходимо, прежде всего, обеспечить получение этим проектом запросов из Интернет. Web-приложение .NET Core содержит встроенный web-сервер Kestrel, который обеспечивает прием пакетов по протоколам http и https на определенный порт и передачу его на дальнейшую обработку приложением. Раньше его использование в качестве пограничного сервера, т.е. шлюза, открытого в Интернет, не рекомендовалось в силу того, что он был недостаточно защищен. В настоящее время этот web-сервер снабжен необ-

ходимыми настройками, позволяющими предотвратить большинство известных DDoS атак, в частности таких, как атака большими пакетами или атака замедлением трафика.

Однако его использование как пограничного сервера имеет ряд недостатков. Основным из них является то, что Kestrel не анализирует параметр запроса 'Host', а обрабатывает весь трафик, приходящий на определенный порт, и передает его тому приложению, в состав которого он входит. Тем самым становится невозможным иметь на одном сервере с определенным IP-адресом несколько различных web сайтов, работающих по стандартному порту, или обрабатывать запросы, в зависимости от их вида, разными приложениями.

Для решения этой проблемы используется стандартный web-сервер, который конфигурируется как обратный прокси-сервер. При такой конфигурации Kestrel конфигурируется на работу с каким-либо внутренним портом, например 5000, а запросы, приходящие из Интернета на стандартные порты, обрабатываются стандартным web-сервером, который настраивается на способ обработки запросов в зависимости от содержащегося в запросе адреса (параметра 'Host'). Те запросы, которые должны обрабатываться конкретным приложением с сервером Kestrel, транслируются ему по протоколу http от имени локального хоста на тот порт, на который данный Kestrel сконфигурирован.

Однако при такой трансляции теряются параметры http заголовка, содержащие адрес клиента и тип протокола, поскольку заменяются локальными значениями.

Для того, чтобы приложение могло корректно получать эти данные, например, для ведения протокола или верификации запросов, необходимо соответствующим образом настроить трансляцию заголовков в прокси-сервере и задать соответствующие параметры в промежуточном слое приложения, отвечающего за конфигурацию обработки протокола http. Ответ от Kestrel возвращается клиентам.

В данном проекте в качестве обратного прокси-сервера был использован web сервер Nginx.

Предполагалось, что основной проблемой переноса системы на другую платформу будет переход на другой SQL сервер. Первоначально система разрабатывалась под использование MS SQL Server. Однако, согласно новым требованиям, SQL сервер было необходимо заменить.

В качестве нового сервера был предложен PostgreSQL. PostgreSQL – это реляционная база данных с открытым кодом, которая поддерживается в течение 30 лет разработки и является одной из наиболее известных среди всех существующих реляционных баз данных. Популярностью у разработчиков и администраторов база данных PostgreSQL обязана своей исключительной гибкости и целостности [11].

Хотя формально язык манипулирования данными SQL и стандартизован, однако в реальности каждая реализация SQL сервера имеет свой диалект этого языка. Так, например, в MS SQL в идентификаторах объектов можно использовать как строчные, так и прописные буквы. В PostgreSQL, если в идентификаторе используются прописные символы, то этот идентификатор необходимо заключить в двойные кавычки.

Поэтому, в общем случае, потребовалось бы значительная переработка кода, генерирующего SQL команды. В случае конкретного проекта ситуация сильно упростилась благодаря использованию для работы с данными пакета Entity Framework – средства, реализованного на языках семейства .NET для работы с базами данных. Этот пакет позволяет взаимодействовать с SQL сервером с помощью сущностей (entity), а не SQL команд.

При таком подходе каждой таблице в базе данных ставится в соответствие соответствующий класс языка программирования, элементы которого соответствуют столбцам таблицы. Конкретная сущность является экземпляром класса с данными, соответствующими данным конкретной строки соответствующей таблицы. Связи между таблицами реализуются включением в соответствующий класс ссылок на классы, соответствующие связанным таблицам.

Выборка сущностей из базы данных осуществляется посредством поискового условия, задаваемого в терминах используемого языка программирования, а не SQL. Изменения в базе данных обеспечиваются изменениями соответствующих сущностей.

Таким образом, работа с данными происходит средствами языка программирования, а не выполнением SQL команд. Поэтому для перехода на работу с другим SQL сервером оказалось достаточным заменить библиотеки, обеспечивающие функциональность Entity Framework, с библиотек MS SQL на библиотеки PostgreSQL.

К сожалению, использование для работы с данными исключительно средств Entity Framework не всегда позволяет обеспечить необходимую функциональность. Это, прежде всего, касается средств конкретного SQL сервера, выходящих за рамки стандартного SQL и не отраженных в средствах Entity Framework. В частности это относится к средствам, обеспечивающим возможности полнотекстового поиска.

Другой проблемой является сложность, а иногда и невозможность динамического построения сложных поисковых условий, особенно со связанными таблицами, которое в данном проекте необходимо для обеспечения функциональности блока универсального поиска ЭБ НИР. Поэтому для обеспечения нужной функциональности в этих случаях в проекте использовалась генерация непосредственно SQL команд. Поскольку, как отмечалось ранее, версии языка SQL в серверах различаются, эти места в проекте пришлось переработать.

Самой сложной проблемой переноса системы на другую платформу оказалась проблема работы с изображениями. В ЭБ НИР накоплен достаточно большой объем отсканированных публикаций, который на момент написания статьи составляет более 5 600 000 страниц, занимающих более 3.60 терабайт дискового пространства.

По методике, принятой при создании ЭБ НИР, сканирование осуществлялось с максимально возможным качеством и, в основном, составляет 600 dpi. Такое высокое качество изображений является излишним для просмотра на экране компьютера, при передаче изображений в неизменном виде, приведет к излишнему трафику и, как следствие – к значительному замедлению работы.

Кроме того, основным форматом хранения изображений в системе является TIFF (Tagged Image File Format) [12].

Этот формат был принят в системе качестве основного, поскольку обеспечивает максимальную компрессию черно-белых текстов при помощи алгоритмов сжатия CCITT Group 3 и 4, а, на период создания системы, вопросы оптимизации использования дискового пространства были чрезвычайно актуальны.

К сожалению, формат TIFF не является стандартным для Интернета, и большинство браузеров его не поддерживает. Поэтому, при запросе пользователем конкретной страницы, она преобразуется в стандартный для Интернета формат PNG (Portable Network Graphics) [13] с масштабированием, которое выбирает пользователь в соответствии со своими нуждами.

Данное преобразование изображений в системе осуществлялось с использованием графической библиотеки Microsoft System.Drawing.Common (MSDC). Эта библиотека по существу является тонкой оболочкой между программой и интерфейсом GDI+ (Graphics Device Interface Plus) [14], который производит обработку изображений в операционных системах Windows и является их частью. Поэтому при переводе системы, использующей графическую библиотеку MSDC, на другую операционную платформу требуется установка на нее дополнительных средств, обеспечивающих функционирование данного интерфейса.

Как выяснилось, единственной библиотекой, реализующей интерфейс GDI+ на Unix, оказалась библиотека libgdiplus [15]. При попытке ее использования выяснилось, что в сканах публикаций “Научного наследия” имеется большое количество TIFF-файлов, при попытке от-

крытия которых библиотекой libgdiplus выдается пустое изображение. При этом программа не получает никаких уведомлений о том, что данный файл был обработан некорректно.

Анализ ситуации показал, что многие программные пакеты, на которых осуществлялось сканирование, не всегда корректно формировали результирующие файлы в формате TIFF. При этом интерфейс GDI+ обрабатывает эти файлы без проблем. Необходимо отметить, что в настоящее время, начиная с версии .NET Core 6.0, Microsoft не рекомендует использовать библиотеку MSDC для кроссплатформенных приложений, мотивируя это ненадежностью библиотеки libgdiplus.

Выходом из этой ситуации стал переход на другую графическую библиотеку. В качестве вариантов рассматривались кроссплатформенные библиотеки ImageSharp и SkiaSharp.

Библиотека SkiaSharp поддерживается Google и обеспечивает высокую производительность при манипулировании с изображениями, однако не поддерживает формат TIFF.

Библиотека ImageSharp [16] является свободно распространяемым продуктом фирмы SixLabors. Она, хотя менее производительна, чем SkiaSharp, но реализована на платформе .NET и включается в проект, а не устанавливается как сервис в операционную систему, и одинаково работает как в среде разработки под Windows, так и в среде выполнения под Unix. Поэтому проект был переведен на этот графический пакет. Как выяснилось, при работе с данной библиотекой, как и с библиотекой libgdiplus, возникают проблемы при открытии некоторых TIFF файлов, но в отличии от последней, библиотека ImageSharp корректно информирует программу о возникшем исключении. Для разрешения ситуации был проведен тотальный анализ архива изображений для того, чтобы выявить файлы, вызывающие проблемы обработки. Эти файлы по протоколу SFTP скачивались из архива на компьютер под управлением Windows, перепаковывались программой с использованием библиотеки MSDC и замещали исходные файлы.

Еще одна проблема, с которой пришлось столкнуться, возникла при переходе к новой версии .NET Core. Первоначально система была реализована на платформе 3.1, которая была объявлена как lts (долговременная поддержка). Однако в настоящее время поддержка этой версии прекращена и возникла необходимость перевода системы на новую версию с опцией lts – версию 6.0. Изменения в коде проекта, связанные с тем, что все библиотеки, использованные в проекте, были обновлены, были минимальны. Неожиданной проблемой стало то, что пакет 'dotnet' версии 6.0, устанавливаемый на Unix, "зависал" при обращении к файловому хранилищу, на котором располагался массив отсканированных публикаций. Поскольку выяснение причин "зависания" могло занять неопределенное время, модуль обработки изображений был вынесен в отдельный web-сервис, который был реализован на Python, программы на котором проблем с доступом к файловому хранилищу не имели. В дальнейшем, в ходе ряда экспериментов было выяснено, что проблемы у 'dotnet' 6.0 вызывает протокол подключения удаленных хранилищ nfs версии 3, который использовался в установленной конфигурации аппаратных средств. При переходе на последнюю версию протокола nfs 4 проблема исчезла.

4. ЗАКЛЮЧЕНИЕ

В результате опыта, полученного при переводе информационной системы, разработанной на Windows, в операционную систему Unix, можно сделать вывод о том, что современные средства кроссплатформенной разработки позволяют достаточно эффективно разрабатывать приложения, функционирующие на разных серверных платформах. При этом переход на другую платформу может потребовать определенных изменений, но они не являются непреодолимыми.

СПИСОК ЛИТЕРАТУРЫ

1. Каленов, Н.Е., Погорелко К.П., Сотников А.Н. О развитии электронной библиотеки “Научное наследие России” как составляющей Единого цифрового пространства научных знаний // Информационные процессы, 2022, Т. 22, № 3, С. 155–166.
2. Каленов Н.Е., Савин Г.И., Сотников А.Н. Технология создания электронной библиотеки “Научное наследие России” // Научная книга на постсоветском пространстве: материалы II Международной научной конференции, Москва, 19–21 сентября 2007 года. Москва: Наука, 2007, С. 11–16.
3. Каленов Н.Е., Савин Г.И., Сотников А.Н. Электронная библиотека “Научное наследие России” // Информационные ресурсы России, 2009, № 2(108), С. 19–20.
4. Нестеренко А.К., Сысоев Т.М., Погорелко К.П. Задача реализации электронной библиотеки "Научное наследие России"как распределенной информационной системы // Новые технологии в информационном обеспечении науки: Сборник научных трудов. М.: Научный мир, 2007, С. 276–287.
5. Якшин М.М. Платформа SciRus – основа технологического комплекса электронной библиотеки “Научное наследие России” // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды XVI Всероссийской научной конференции RCDL-2014. Дубна: ОИЯИ, 2014, С. 362–368.
6. Бездушный А.А., Бездушный А.Н., Серебряков В.А., Филиппов В.И. Интеграция метаданных Единого Научного Информационного Пространства РАН. М.: ВЦ РАН, 2005, 238 с.
7. Погорелко, К.П. Новая система презентации электронных книг в системе “Научное наследие России” // Информационное обеспечение науки: новые технологии: Сборник научных трудов. Москва: БЕН РАН, 2013, С. 32–35.
8. Погорелко, К.П. Новая версия программного обеспечения электронной библиотеки “Научное наследие России” // Информационные ресурсы России, 2020. № 5(177), С. 27–29.
9. https://learn.microsoft.com/ru-ru/dotnet/core/introduction
10. https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide
11. https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-postgresql/
12. https://www.rfc-editor.org/rfc/rfc3302.txt
13. http://www.libpng.org/pub/png/
14. https://learn.microsoft.com/ru-ru/windows/win32/gdiplus/-gdiplus-about-gdi–about
15. https://www.mono-project.com/docs/gui/libgdiplus/
16. https://sixlabors.com/products/imagesharp/

About some aspects of porting Windows web applications to Unix environment

K. Pogorelko

The article deals with the issues of porting a web application implemented in the .NET Core cross-platform system from the Windows operating system to the Unix operating system. The aspects of application deployment, migration to another SQL server and problems with image processing libraries are considered. It is concluded that the use of cross-platform systems for application development makes it possible to effectively develop applications that operate on different server platforms, however, transferring the system to another server platform can cause certain problems.

KEYWORDS: information systems, web applications, .NET Core, cross-platform systems, portability issues, image processing