

Пакет с NuGet-пакетами в GitLab: от общих DLL к внутреннему NuGet Repository

А у вас есть пакет с пакетами?

Почти в каждом доме есть пакет с пакетами. На первый взгляд странная традиция, но все знают, куда складывать пакеты, которые ещё пригодятся. С NuGet-пакетами у нас получилось примерно так же.

К своему **пакету с пакетами** мы пришли не сразу. Плагины для Revit, в том числе созданные разными компаниями, загружаются в один процесс и могут принести разные версии одних и тех же DLL. Какая сборка загрузится первой, та и может определить поведение остальных. Иногда плагин не загружается сразу, а иногда ошибка может проявиться уже в рантайме.

От DLL на сетевом диске к NuGet-пакетам

Первые конфликты у нас возникли вокруг UI-библиотек **Prism**, **Microsoft.Xaml.Behaviors** и **MaterialDesign**. Для них сделали форки с префиксом неймспейсов **MarksDigital**.

Первоначально они жили в одной репозитории, а собранные DLL лежали на сетевом диске.

Время от времени в форки требовалось вносить отдельные правки, и схема с общей папкой DLL становилась всё менее удобной. **MarksDigital.Prism**, **MarksDigital.Microsoft.Xaml.Behaviors** и **MarksDigital.MaterialDesign** переехали в отдельные репозитории, в каждом настроили сборку NuGet-пакета. Сначала пакеты собирались и публиковались локально батником, позже тот же процесс перенесли в GitLab CI. Версия пакета берётся из Git-тега, а сборка и публикация выполняются на runner.

Почему GitLab

К тому моменту GitLab уже использовался для исходного кода, а встроенный NuGet Package Registry позволял обойтись без ещё одного сервера. Не нужно было разворачивать, обновлять и резервировать отдельную систему хранения пакетов. Права доступа, проекты, runner и резервное копирование уже были частью имеющейся инфраструктуры.

Первоначально в репозитории находились только наши пакеты. Сторонние зависимости восстанавливались напрямую с nuget.org, и версии пакетов в плагинах постепенно начали расходиться. Аудит и выравнивание версий исправляли ситуацию только на конкретный момент: уже на следующий день в одном из проектов могла появиться другая версия или новая библиотека.

Поэтому в GitLab стали переносить и сторонние пакеты. Перед публикацией проверялись конкретная версия, её лицензия и транзитивные зависимости. GitLab Registry сам не проверяет лицензии — это делается до помещения пакета во внутренний репозиторий.

Когда версия меняет лицензию

Open source даже для невирусных лицензий не означает "бесплатно для любых целей и навсегда": условия могут меняться между версиями. Примером может служить пакет

Extended.Wpf.Toolkit, содержащий сборку **Xceed.Wpf.Toolkit.dll**.

В README [официального репозитория](#) Xceed указывает, что Extended WPF Toolkit распространяется по Xceed Community License начиная с версии 4.0.0. Однако история репозитория и опубликованные NuGet-пакеты показывают более сложную картину:

- [3.6.0](#) - последняя версия, в которой репозиторий и NuGet-пакет согласованно указывают открытую лицензию MS-PL;
- [3.7.0](#) - тег в Git еще содержит MS-PL, но опубликованный пакет уже снабжён Xceed Community License с ограничением на коммерческое использование. Сразу после создания тега лицензию заменили и в [репозитории](#);
- [3.8.0](#) и [3.8.1](#) также опубликованы по Xceed Community License;

- **3.8.2** снова объявлена пакетом под MS-PL. В release notes прямо сказано, что это эквивалент 3.8.1 с лицензией MS-PL. По странице nuget.org и метаданным пакет может выглядеть подходящим для использования, но внутри архива остался файл LICENSE.txt с текстом Community License. Это неочевидный риск: проверка только метаданных не покажет противоречие в самом пакете;
- **4.0.0** распространяется по Xceed Community License и соответствует формулировке из README.

Первые три из перечисленных версий - 3.6.0, 3.7.0 и 3.8.0 - сейчас помечены на nuget.org как delisted. Это не означает, что пакеты удалены: по прямой ссылке остаются доступны страница версии и её метаданные, а restore по точно заданной версии продолжает работать. Такие версии скрыты из обычного поиска и общего списка версий.

Последняя версия без таких противоречий - 3.6.0 распространявшаяся по MS-PL.

Пакеты тянут за собой пакеты

Перенести во внутренний репозиторий только нужный прямой пакет недостаточно. У него есть транзитивные зависимости, у них - свои и так далее. Если хотя бы одного пакета из этой цепочки нет внутри, изолированный restore падает.

Сначала недостающие пакеты выявлялись по ошибкам сборки: загрузил, опубликовал, запустил ещё раз.

Потом проект стали восстанавливать на компьютере с доступом к nuget.org: NuGet автоматически загружал в локальный кэш как прямые, так и все транзитивные зависимости. Их всё равно приходилось вручную собирать из кэша и публиковать во внутренний репозиторий. Чтобы не собирать пакеты из кэша вручную, мы разработали утилиту **MarksDigital.NuGetUtils**. Она загружает заданный пакет вместе с транзитивными зависимостями с nuget.org, а затем сверяет подготовленный набор со внутренним репозиторием, чтобы не публиковать повторно то, что уже есть.

Лицензии утилита не проверяет. Решение о публикации принимает человек, ответственный за допуск пакетов во внутренний репозиторий.

Позже мы посмотрели, можно ли заменить утилиту готовым сервером пакетов. Например, [BaGet](#) и [Nexus Repository Community Edition](#) умеют работать как прокси к nuget.org: во время restore NuGet запрашивает прямые и транзитивные зависимости, а сервер сохраняет их локально.

Но для нас в такой схеме теряется важная граница: новый пакет попадает во внутренний контур до того, как ответственный проверит конкретную версию и её лицензию.

Поэтому мы планируем развивать **MarksDigital.NuGetUtils** как инструмент подготовки пакетов к допуску.

Внутренний репозиторий как единственный источник

На GitLab Runner источник nuget.org отключён, а сетевой доступ к нему закрыт. Если разработчик добавит несогласованную зависимость, чистая сборка не сможет её восстановить и restore завершится ошибкой.

Версии прямых зависимостей вынесли в общие **.props**-файлы. Внутренний репозиторий определял, какие пакеты доступны, а **.props**-файлы - какие именно версии используются в проектах.

Когда nuget.org оказался недоступен из-за сбоя или сетевых ограничений, наши сборки продолжили работать с внутренними копиями пакетов.

Что в итоге

К моменту подготовки плагинов к выходу на рынок во внутреннем репозитории уже были конкретные проверенные версии пакетов. Не пришлось в авральном режиме выяснять, какие библиотеки входят в поставку, разбирать конфликты версий и с нуля собирать сведения о сторонних лицензиях.

Так у нас и появился свой **пакет с пакетами**: мы точно знаем, какая версия лежит внутри и почему она там оказалась.