


Реактивная сборка

Огромного проекта

Станислав Сидристый

- Системный архитектор в Центре Речевых Технологий
- Книга:  <https://github.com/sidristij/dotnetbook>
- telegram: @sidristij
- sunex.development@gmail.com

Что дано?

- монорепозитарий
- более 100 файлов решений
- более 1000 проектов
- выпуск идёт под общим срезом
- отдельные билды под каждый сервис



- есть общие nuget-пакеты
- есть общие проекты (в рамках группы сервисов)



- есть общие nuget-пакеты
- есть общие проекты (в рамках группы сервисов)

Первичная проблема

- трудно отслеживать что было изменено (nuget+сервисы)
- трудно отслеживать что нужно пересобрать (nuget+сервисы)
- версии сервисов должны расти только если они были изменены
- dev/release версии

Так что же делать?

Можно попробовать сделать
обычным .NET путём

- прошлый билд
- *изменения*
- текущий билд
 - msbuild сам всё понимает по датам изменения файлов

Проблемы

- git не сохраняет даты изменения файлов

Решение

```
find . -exec touch -m -d '-1 day' {} +  
/// достать откуда-то bin/Release прошлого билда  
git diff --name-only PREV-SHA HEAD | xargs -d'\n' touch -m -d  
dotnet build -c Release
```

Проблемы

- msbuild нужен не только bin/Release, но и **obj/Release**
- их объёмы будут под 20Gb на полную сборку
- из-за больших объемов билд будет не супер-быстрым

Можно попробовать сверять
bin/Release

Решение

Dockerfile билда:

```
FROM registry/<service>:<version> AS prev-version
```

```
FROM dotnetsdk as build
```

```
...
```

```
+ COPY --from=prev-version app <sources-path>/last
```

```
dotnet build -c Release
```

```
diff /last /bin/Release/net6.0
```

Проблемы

- из-за многопоточной сборки dll отличаются
- если отключить многопоточную сборку, будет очень долго

Можно попробовать вручную

Гипотеза

- Состояние любого проекта зависит от:
 - исходников проекта;
 - зависимостей проекта:
 - других проектов;
 - nuget пакетов;
 - *.dll, *.so;
 - Contents.

Гипотеза

Чтобы рассчитать «версию» проекта надо:

- рассчитать число, которое определит версию исходников и contents проекта
- рассчитать число, которое определит «версии» всех зависимостей

Версии

dev

feature

34.45364

34.45374

HASH₁



34.45375

34.45378

HASH₂



34.45381

HASH₃



34.45384

Версии

HASH_1

34.45364



34.45374

34.45375

1. Рассчитать HASH-версию сервиса

HASH_2

2. Забрать полный список тэгов



34.45378

3. Забрать максимальную версию

HASH_3

34.45381



34.45384

Версии

HASH_1

34.45364



34.45374

34.45375

1. Рассчитать HASH-версию сервиса

HASH_2

2. Забрать полный список тэгов



34.45378

3. Забрать максимальную версию

HASH_3

34.45381



34.45384

Как считать hash-версию?

1. Если msbuild запустить на каком-либо проекте, тот сначала соберёт все зависимые проекты, а потом – требуемый;
2. У msbuild есть язык описания шагов сборки.

Что получим

1. Не будет компиляции если она не нужна
2. Меняем версию только если что-то изменилось

План

1. Пишем msbuild Task для подсчёта hash-версии;
2. Пишем ещё один для обхода проектов.

```
C# CollectHashVersions.cs x
1  > using ...
9  // ReSharper disable UnusedAutoPropertyAccessor.Global
10
11 namespace MsBuildMarkChanged
12 {
13     /// <summary>
14     /// Common properties: https://learn.microsoft.com/en-us/visualstudio/msbuild/common-msbuild-project-properties?view=vs-2022
15     /// </summary>
16     public class CollectHashVersions : Task
17     {
18         private const string MetaDataWarn = "{0} doesn't contain {1} and wasn't used in assembly hash calculation";
19
20         private List<(string name, string version)> _packages;
21         private ITaskItem[] _references;
22         private ITaskItem[] _contents;
23         private string _fullHashSet;
24         private MD5 _md5;
25         private string _outputPath;
26         private string _projectPath;
27
28         public CollectHashVersions()
29         {
30             _md5 = MD5.Create();
31         }
32
33         /// <summary>
34         /// @(Compile)
35         /// </summary>
36         [Required]
37         public ITaskItem[] CompileItems { get; set; }
38
39         [Required]
40         public string ProjectPath
41         {
42             get => _projectPath;
43             set => _projectPath = value.NormalizePath();
44         }
45
46         [Required]
47         public string OutputPath
48         {
49             get => _outputPath;
50             set => _outputPath = value.NormalizePath();
51         }
52     }
53 }
```

Выводы

Выводы

- Вместо линейных 20-25 минут линейные 20 сек – 3 минуты;
- Вместо 400-500 машиноминут 20 секунд – 3 минуты;
- Снижение количества ошибок «забыли собрать puget»;
- Снижение количества ошибок «забыли собрать сервис»;
- Автоматический деплой среза версий сервисов;
- Версия растёт только если есть изменения;
- Можно удостовериться, что изменения затронули мало сервисов.

QA