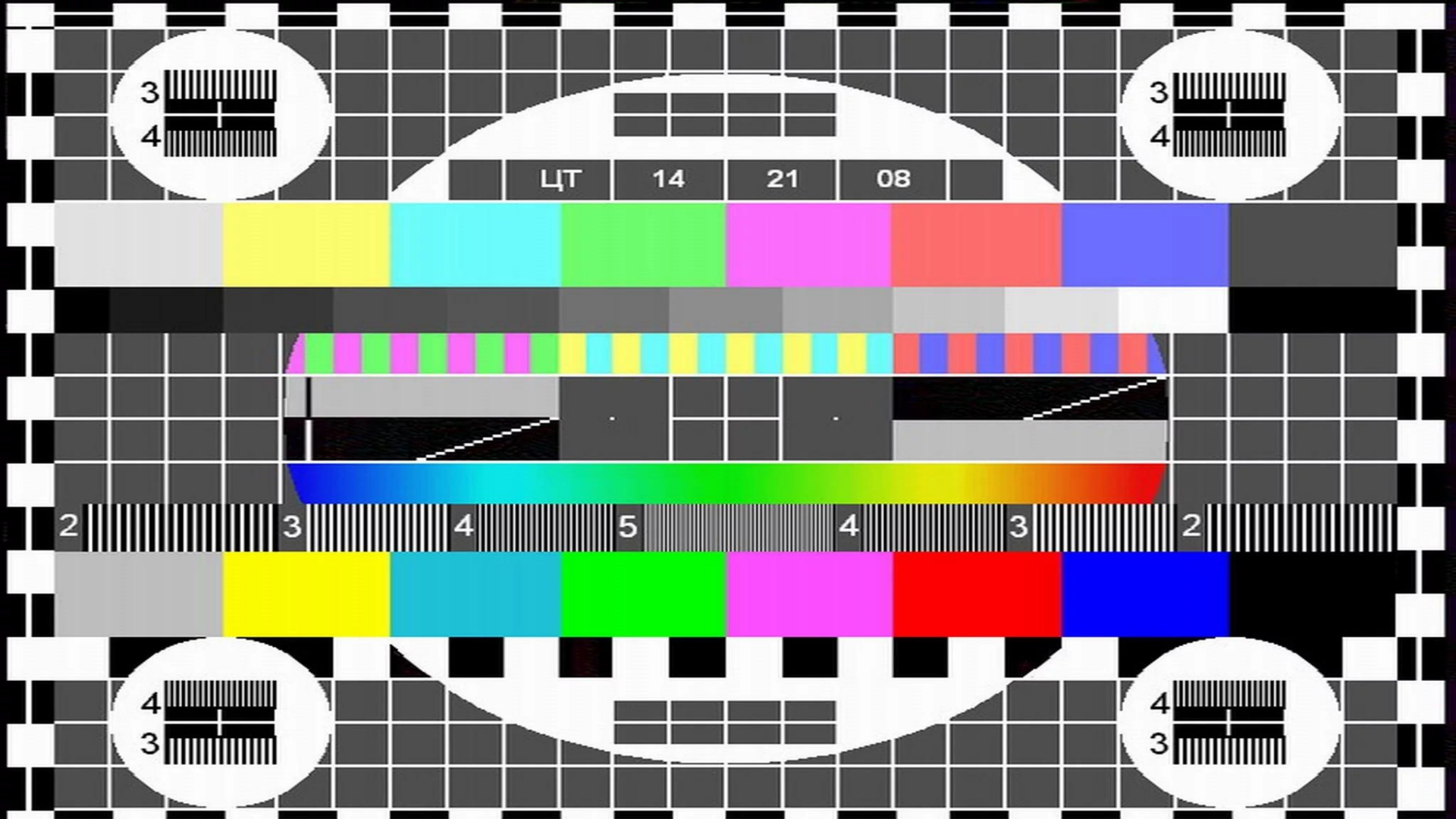
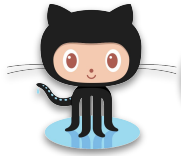

3
4
ЦТ 14 21 08
3
4
2 3 4 5 4 3 2
4
3
4
3

C#/CI/CD

Как собирать проекты просто

Пьяников Николай



GitHub /NikolayPianikov

Сценарии сборки проектов в .NET

В среде разработки (IDE)

- Отладка, тестирование, запуск

Из командной строки

- Автоматизация сценариев, CI/CD

MSBuild

Свойства

Группы

Цели

Задачи

```
<Project DefaultTargets="Compile">
    <!-- Set the application name as a property -->
    <PropertyGroup>
        <appname>HelloWorldCS</appname>
    </PropertyGroup>
    <!-- Specify the inputs by type and file name -->
    <ItemGroup>
        <CSFile Include="*.cs"/>
    </ItemGroup>
    <Target Name="Compile">
        <!-- Run the C# compilation using input files of type CSFile -->
        <CSC Sources="@(CSFile)" OutputAssembly="$(appname).exe">
            <!-- Set the OutputAssembly attribute of the CSC task
            to the name of the executable file that is created -->
            <Output TaskParameter="OutputAssembly" ItemName="EXEFile"/>
        </CSC>
        <!-- Log the file name of the output file -->
        <Message Text="The output file is @(EXEFile)"/>
    </Target>
</Project>
```

Инструменты сборки на основе разметки

MSBuild

NAnt

Apache Maven

Apache Ant

Apache Ivy

…

Достоинства

Синтаксис хорошо структурирован

Простая модель

- свойства
- элементы
- задачи
- цели

Высокий уровень интеграции с IDE

Хорошо работают в стандартных сценариях

Недостатки

Ограничения модели проекта

“Слабая выразительность” языков разметки

При реализации нестандартных сценариев

- Большие трудозатраты
- Дорогая поддержка

Сборка с использованием скриптов автоматизации

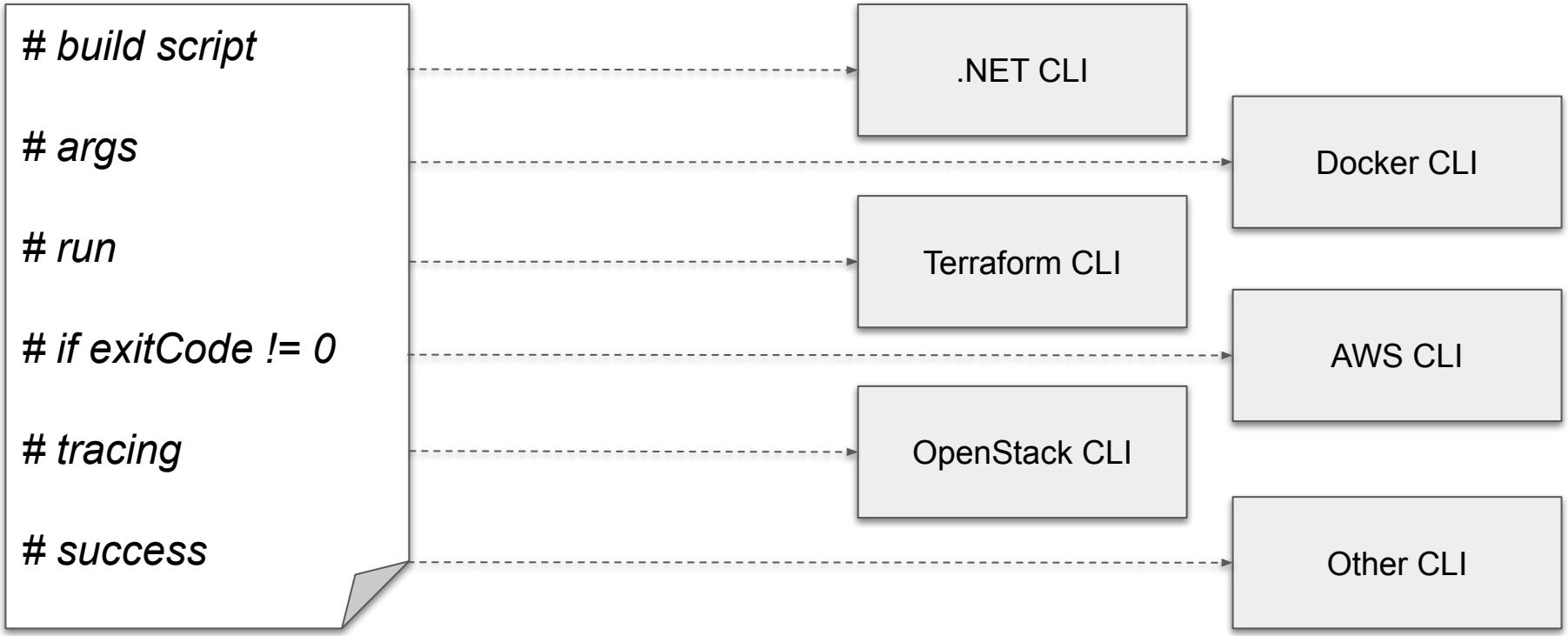

Bash

Windows command shell


PowerShell

…

Сборка с использованием скриптов автоматизации



Достоинства

Автоматизация сложных и нестандартных сценариев

Недостатки

Необходимость изучать язык сценариев

Слабая структурированность кода

Зависимость от платформы

Сложность отладки

“Примитивный” API для взаимодействия с CLI

- Аргументы командной строки
- Не структурированный вывод на консоль StdOut/StdErr
- Код выхода

Сборка с использованием высокоуровневых языков

Cake для .NET на C# script и C#

Nuke для .NET на C#

Gradle Build Tool для JVM на Groovy и Kotlin

Rake (Ruby Make) для Ruby на Ruby

Bazel для разных платформ на Starlark

и другие

Cake

```
var target = Argument("target", "Test");
var configuration = Argument("configuration", "Release");
var solution = "../CI-CD.sln";

Run Task
Task("Build").Does(() =>
{
    DotNetBuild(solution,
        new DotNetBuildSettings { Configuration = configuration, NoLogo = true });
});

Run Task
Task("Test").IsDependentOn("Build").Does(() =>
{
    DotNetTest(solution,
        new DotNetTestSettings { Configuration = configuration, NoLogo = true, NoBuild = true });
});

RunTarget(target);
```

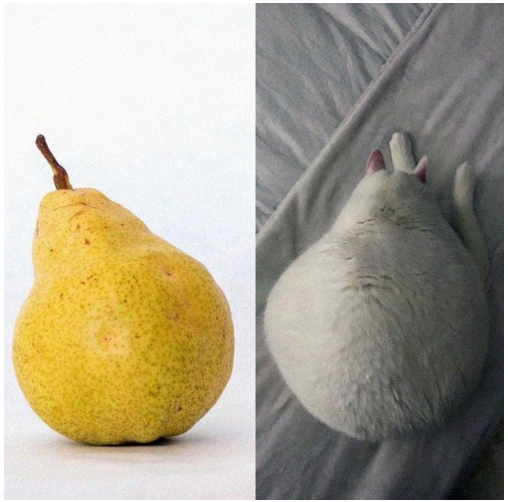
Cake Frosting

```
new CakeHost().UseContext<BuildContext>().Run(args);

public class BuildContext(ICakeContext context)
    : FrostingContext(context)
{
    public const string Solution = "../CI-CD.sln";
    public string BuildConfiguration { get; } = context.Argument("configuration", "Release");
}

[TaskName("Build")]
public sealed class BuildTask : FrostingTask<BuildContext>
{
    public override void Run(BuildContext context) => context.DotNetBuild(BuildContext.Solution,
        new DotNetBuildSettings { Configuration = context.BuildConfiguration, NoLogo = true });
}

[TaskName("Test"), IsDependentOn(typeof(BuildTask))]
public sealed class TestTask : FrostingTask<BuildContext>
{
    public override void Run(BuildContext context) => context.DotNetTest(BuildContext.Solution,
        new DotNetTestSettings { Configuration = context.BuildConfiguration, NoLogo = true, NoBuild = true });
}
```



Nuke

```
[TypeConverter(typeof(TypeConverter<Configuration>))]
public class Configuration : Enumeration
{
    public static Configuration Debug = new() { Value = nameof(Debug) };
    public static Configuration Release = new() { Value = nameof(Release) };
    public static implicit operator string(Configuration configuration) => configuration.Value;
}

class BuildSolution : NukeBuild
{
    public static int Main () => Execute<BuildSolution>(x => x.Test);

    [Parameter("Configuration to build")]
    readonly Configuration Configuration = IsLocalBuild ? Configuration.Debug : Configuration.Release;

    Target Build => _ => _
        .Executes(() => DotNetTasks.DotNetBuild(
            new DotNetBuildSettings().SetConfiguration(Configuration).SetNoLogo(true)));

    Target Test => _ => _ .DependsOn(Build)
        .Executes(() => DotNetTasks.DotNetTest(
            new DotNetTestSettings().SetConfiguration(Configuration).SetNoLogo(true).SetNoBuild(true)));
}
```

Достоинства

Автоматизация нестандартных сценариев

Знакомый язык

Возможность отладки

Независимость от платформы

API для взаимодействия с CLI

- Набор команд и их аргументы

Недостатки

Ограничения модели (Task, Target, DependsOn …)

Слабый API для сборки .NET проектов

- Не структурированный вывод
- Код выхода

Идеальный инструмент для сборки

Автоматизация нестандартных сценариев

Знакомый язык

Независимость от платформы

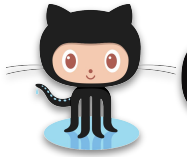
Возможность отладки

Нет ограничений на модель сборки

Привычные библиотеки и практики

Мощный API для сборки .NET проектов

C# interactive: система автоматизации сборки для .NET



GitHub/DevTeam/csharp-interactive

/JetBrains/teamcity-csharp-interactive

Режимы работы

1. Интерактивный
2. Выполнение C# скриптов .csx
3. .NET проект сборки

1. Интерактивный - REPL

> dotnet tool install dotnet-csi -g

> dotnet csi

```
Ctrl-C - Exit the REPL.
#help  - Display help on available commands and key bindings.
> using HostApi;
> new DotNetBuild().Build();
From module C:\Program Files\dotnet\dotnet.exe
  Determining projects to restore...
  Restored D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib\MySampleLib.csproj (in 112 ms).
  Restored D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib.Tests\MySampleLib.Tests.csproj (in 330 ms).
D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib\MySampleLib.cs(6,17): warning CS0169: The field 'Calculator._state' is never used
  MySampleLib -> D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib\bin\Debug\net7.0\MySampleLib.dll
D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib\MySampleLib.cs(6,17): warning CS0169: The field 'Calculator._state' is never used
  MySampleLib -> D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib\bin\Debug\net8.0\MySampleLib.dll
  MySampleLib.Tests -> D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib.Tests\bin\Debug\net7.0\MySampleLib.Tests.dll
  MySampleLib.Tests -> D:\Projects\csharp-interactive\Samples\MySampleLib\MySampleLib.Tests\bin\Debug\net8.0\MySampleLib.Tests.dll
> 
```

1. Интерактивный - интеграция в IDE

2. Выполнение C# скриптов .csx

hello.csx

```
WriteLine($"Hello, {Args[0]}!!!");
```

> dotnet csi hello.csx World

```
Hello, World!!!

Running succeeded.
```

3. .NET проект сборки

CSharpInteractive.Templates

> dotnet new install CSharpInteractive.Templates

> dotnet new build

3. .NET проект сборки

Проект консольного приложения .NET

2 точки входа:

- Program.csx

  > *dotnet csi Program.csx*

- Program.cs

  > *dotnet run*

3. .NET проект сборки - Program.csx

```
// To add a reference to the NuGet package:
// #r "nuget: MyPackage, 1.2.3"

// To include code from the file
// in the order in which it should be executed:
// #load "MyClass.cs"

#load "Program.cs"
```

> dotnet csi Program.csx

3. .NET проект сборки - Program.cs

```
using HostApi;

// Build a dotnet solution or project
new DotNetBuild().Build().EnsureSuccess();
```

3. .NET проект сборки - файл проекта

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net8.0</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="CSharpInteractive" Version="1.1.1" />
    </ItemGroup>

</Project>
```

API

- Вывод, логирование и трассировка
- Аргументы и параметры
- Разрешение зависимостей Microsoft DI
- NuGet
- Командная строка
- Docker CLI
- .NET CLI

Вывод, логирование и трассировка

```
// Output API
WriteLine("Hello");
WriteLine("Hello !!!", Color.Highlighted);
Error("Error details", "ErrorId");
Warning("Warning");
Info("Some info");
Trace("Trace message");
```

Аргументы и параметры

```
Info("First argument: " + (Args.Count > 0 ? Args[0] : "empty"));
Info("Version: " + Props.Get("version", "1.0.0"));
Props["version"] = "1.0.1";

var configuration = Props.Get("configuration", "Release");
Info($"Configuration: {configuration}");
```

Разрешение зависимостей Microsoft DI

```
var myServiceProvider = GetService<IServiceCollection>()
    .AddSingleton<MyTarget>()
    .BuildServiceProvider();

var myTarget = myServiceProvider.GetRequiredService<MyTarget>();
myTarget.DoSomething();

class MyTarget(INuGet nuGet, IBuildRunner buildRunner)
{
    public void DoSomething() =>
        buildRunner.Build(new DotNetBuild()).EnsureSuccess();
}
```

NuGet

```
var settings = new NuGetRestoreSettings("MySampleLib")
    .WithVersionRange(VersionRange.Parse("[1.0.14, 1.1)"))
    .WithTargetFrameworkMoniker("net6.0")
    .WithNoCache(true)
    .WithPackageType(NuGetPackageType.Package)
    .WithPackagesPath(".packages");

var packages = GetService<INuGet>().Restore(settings);
foreach (var package in packages)
{
    Info($"{package.Path}: {package.Sha512}, {package.Files.Count}");
}
```

Командная строка

```
var cmd = new CommandLine("whoami")
    .WithArgs("/all");

var result = cmd.Run().EnsureSuccess()
Info(result.State.ToString());

// Asynchronous way
await cmd.RunAsync().EnsureSuccess();
```

Командная строка: сервис

```
public interface ICommandLineRunner
{
    ICommandLineResult Run(
        ICommandLine commandLine,
        Action<Output>? handler = default,
        TimeSpan timeout = default);

    Task<ICommandLineResult> RunAsync(
        ICommandLine commandLine,
        Action<Output>? handler = default,
        CancellationToken cancellationToken = default);
}
```

Командная строка: событие

```
public record Output(
    IStartInfo StartInfo,
    bool IsError,
    string Line,
    int ProcessId);
```

Командная строка: результат

```
public interface ICommandLineResult
{
    IStartInfo StartInfo { get; }
    ProcessState State { get; }
    long ElapsedMilliseconds { get; }
    int? ExitCode { get; }
    Exception? Error { get; }
}
```

Docker CLI

```
var cmd = new CommandLine("whoami");

await new DockerRun("ubuntu")
    .WithCommandLine(cmd)
    .WithAutoRemove(true)
    .RunAsync()
    .EnsureSuccess();
```

API для .NET CLI

```
new DotNetBuild()
    .Build().EnsureSuccess();

// Asynchronous way
await new DotNetTest()
    .WithNoBuild(true)
    .BuildAsync().EnsureSuccess();
```

.NET CLI

- DotNetBuild
- DotNetClean
- DotNetPack
- DotNetRun
- DotNetTest
- MSBuild
- VSTest
- …
- DotNetCustom

```
public partial record DotNetTest(
    IEnumerable<(string name, string value)> Props, IEnumerable<string> Args,
    IEnumerable<(string name, string value)> Vars,
    IEnumerable<(string name, string value)> RunSettings,
    IEnumerable<string> Loggers,
    string ExecutablePath, string WorkingDirectory,
    string Project, string Settings, bool? ListTests, string Filter,
    string TestAdapterPath, string Configuration,
    string Framework, string Runtime, string Output, string Diag,
    bool? NoBuild, string ResultsDirectory, string Collect, bool? Blame,
    bool? BlameCrash, string BlameCrashDumpType,
    bool? BlameCrashCollectAlways, bool? BlameHang,
    string BlameHangDumpType, TimeSpan? BlameHangTimeout,
    bool? NoLogo, bool? NoRestore, string Arch, string OS,
    DotNetVerbosity? Verbosity, string ShortName)
```

.NET CLI: сервис

```
public interface IBuildRunner
{
    IBuildResult Build(
        ICommandLine commandLine,
        Action<BuildMessage>? handler = default,
        TimeSpan timeout = default);

    Task<IBuildResult> BuildAsync(
        ICommandLine commandLine,
        Action<BuildMessage>? handler = default,
        CancellationToken cancellationToken = default);
}
```

.NET CLI: события

```
public record BuildMessage(
    Output Output, BuildMessageState State,
    IServiceMessage? ServiceMessage,
    string Text, string ErrorDetails,
    string Code, string File, string Subcategory,
    string ProjectFile, string SenderName,
    int? ColumnNumber, int? EndColumnNumber,
    int? LineNumber, int? EndLineNumber,
    DotNetMessageImportance? Importance,
    TestResult? TestResult);
```

.NET CLI: события

```
public readonly record struct TestResult(
    TestState State, string Name, string FlowId,
    string SuiteName, string FullyQualifiedName,
    string DisplayName, string ResultDisplayName,
    string Message, string Details,
    TimeSpan Duration, IReadOnlyList<Output> Output,
    string Source, string CodeFilePath, Guid Id,
    Uri? ExecutorUri, int? LineNumber);
```

.NET CLI: результат

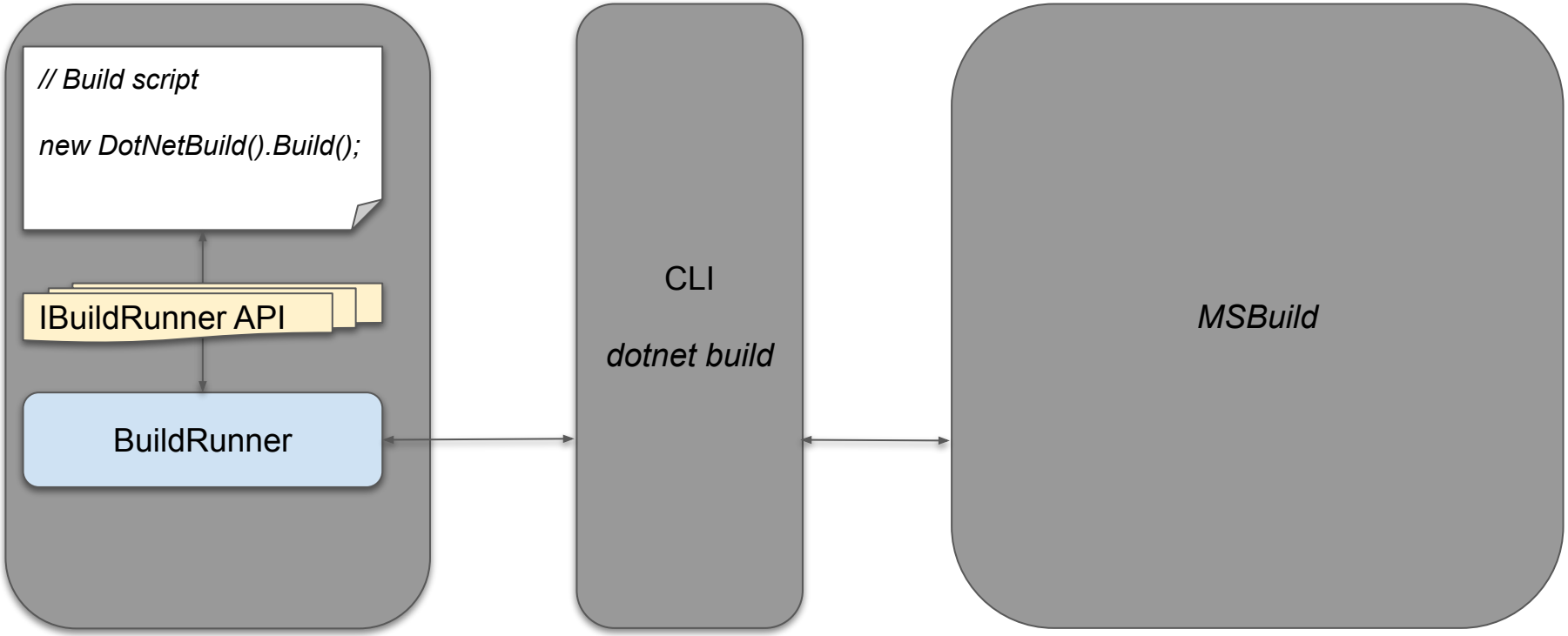
```
public interface IBuildResult: ICommandLineResult
{
    IReadOnlyList<BuildMessage> Errors { get; }
    IReadOnlyList<BuildMessage> Warnings { get; }
    IReadOnlyList<TestResult> Tests { get; }
    BuildStatistics Summary { get; }
}
```

Интеграция с MSBuild

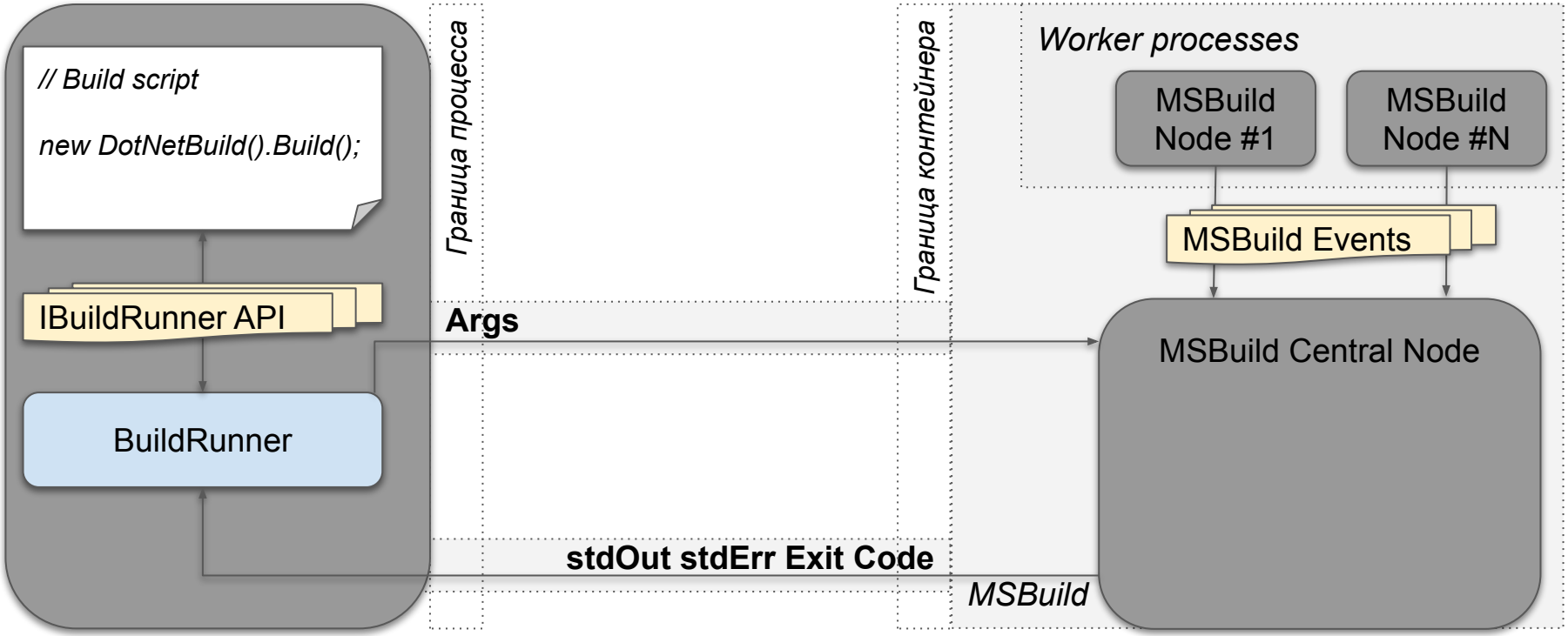
/JetBrains/teamcity-msbuild-logger

TeamCity.Dotnet.Integration

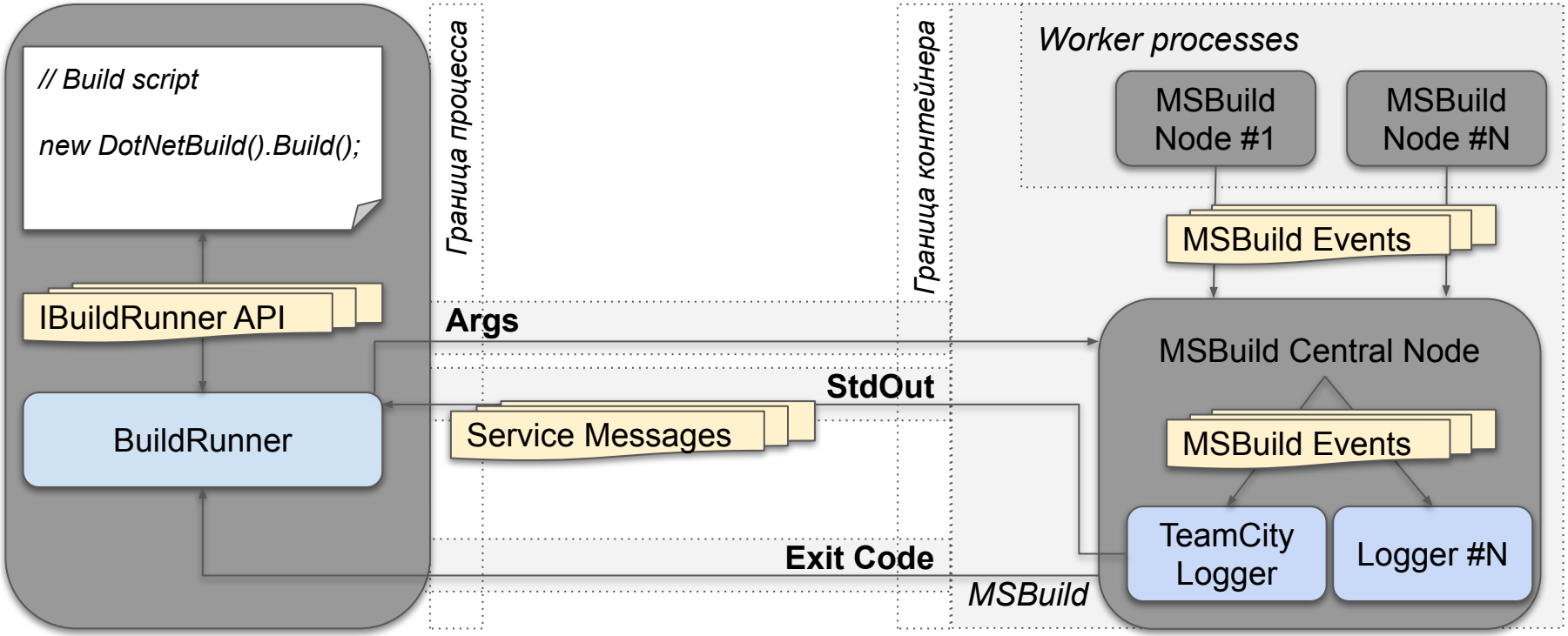
Интеграция с MSBuild



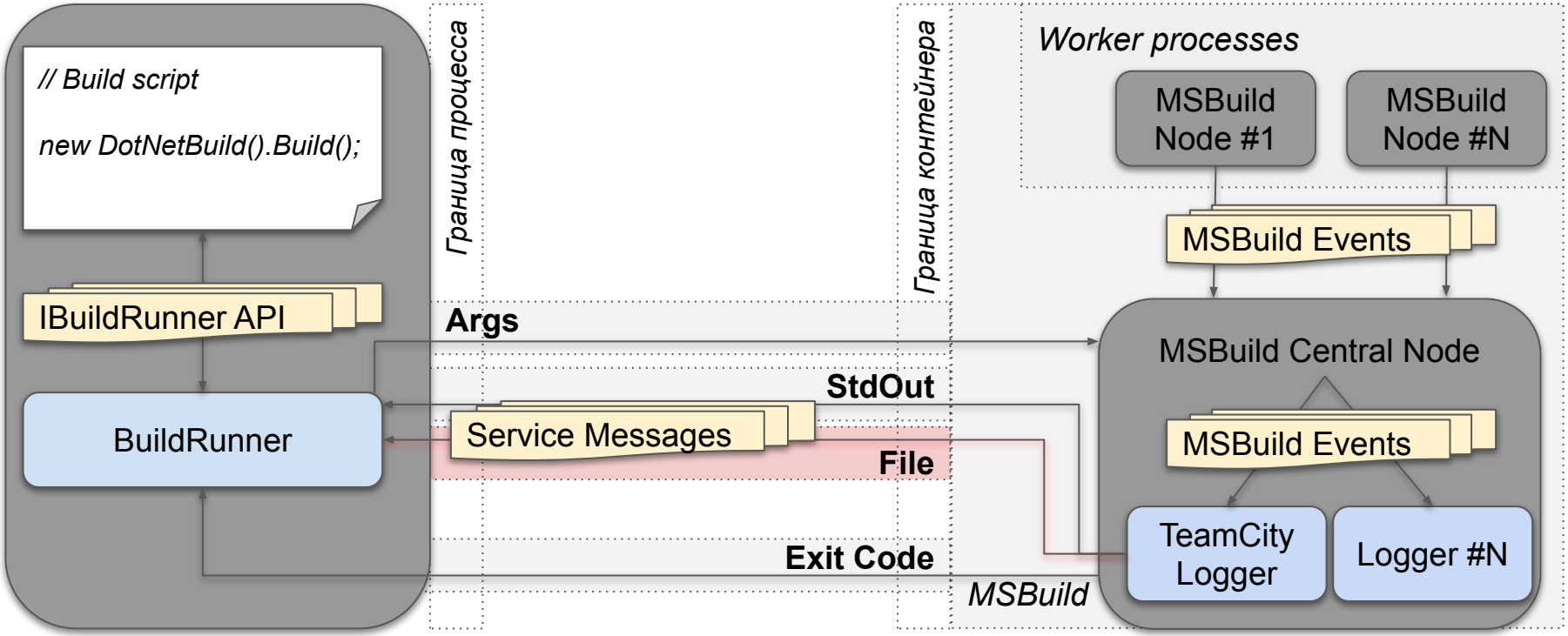
Интеграция с MSBuild



Интеграция с MSBuild



Интеграция с MSBuild

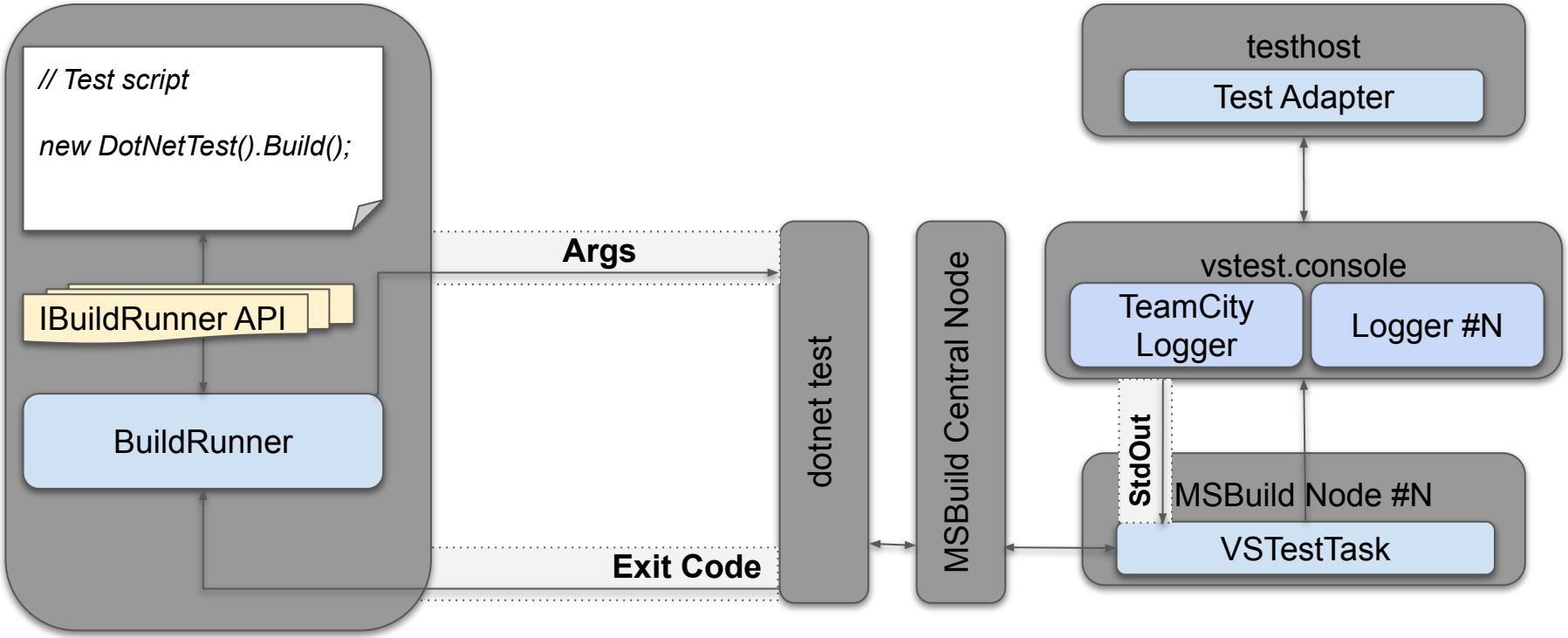


Интеграция с Visual Studio Test Platform (VSTest)

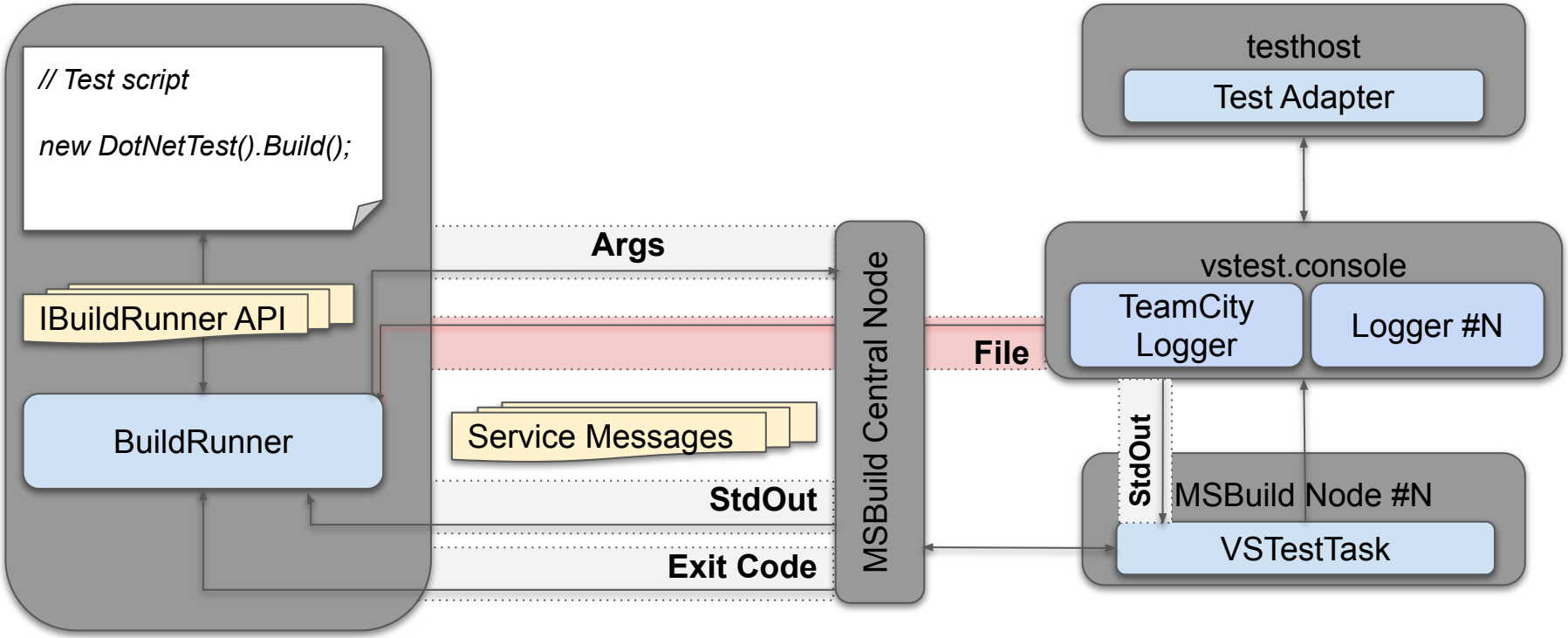
/JetBrains/TeamCity.VSTest.TestAdapter

TeamCity.VSTest.TestAdapter

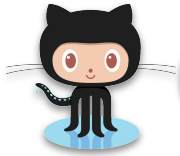
Интеграция с VSTest



Интеграция с VSTest



Примеры



GitHub/DevTeam/ci-cd

Пример сборки

```
var configuration = Props.Get("configuration", "Release");

new DotNetBuild()
    .WithConfiguration(configuration).WithNoLogo(true)
    .Build().EnsureSuccess();

new DotNetTest()
    .WithConfiguration(configuration).WithNoLogo(true).WithNoBuild(true)
    .Build().EnsureSuccess();
```

Использование результатов сборки

```
var buildResult = new DotNetBuild().WithConfiguration(configuration).WithNoLogo(true)
    .Build().EnsureSuccess();

var warnings = buildResult.Warnings
    .Where(warn => Path.GetFileName(warn.File) == "Calculator.cs")
    .Select(warn => $"{warn.Code}({warn.LineNumber}:{warn.ColumnNumber})")
    .Distinct();

foreach (var warning in warnings)
    await new HttpClient().GetAsync(
        "https://api.telegram.org/bot7102686717:AAEHw7HZinme_5kfIRV7TwXK4Xql9WPPpM3/"
        + "sendMessage?chat_id=878745093&text="
        + HttpUtility.UrlEncode(warning));
```

Использование событий сборки

```
var cts = new CancellationTokenSource();
await new DotNetTest()
    .WithConfiguration(configuration)
    .WithNoLogo(true).WithNoBuild(true)
    .BuildAsync(CancellationOnFirstFailedTest, cts.Token)
    .EnsureSuccess();

void CancellationOnFirstFailedTest(BuildMessage message)
{
    if (message.TestResult is { State: TestState.Failed }) cts.Cancel();
}
```

Обобщенная статистика сборки

```
Summary:
  33608 "dotnet build" finished (in 2871 ms) with exit code 0.
  33716 "dotnet test" canceled (in 7095 ms).
  Failed MySampleLib.Tests: MySampleLib.Tests.CalculatorTests.ShouldSub(op1: 3, op2: -2, expected: 5)
    Assert.Equal() Failure: Values differ
    Expected: 5
    Actual:   1
  Failed MySampleLib.Tests: MySampleLib.Tests.CalculatorTests.ShouldSub(op1: 3, op2: 2, expected: 1)
    Assert.Equal() Failure: Values differ
    Expected: 1
    Actual:   5
  D:\Projects\CI-CD\MySampleLib\Calculator.cs(5,17): warning CS0169: The field 'Calculator._state' is never used
  D:\Projects\CI-CD\MySampleLib\Calculator.cs(5,17): warning CS0169: The field 'Calculator._state' is never used
  "dotnet test" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
2 Warning(s)
1 Error(s)
Time Elapsed 0:00:10,5703148
Running FAILED.
```

Сборка

```
var results = await Task.WhenAll(
    RunTestsAsync("7.0", "bookworm-slim", "alpine"),
    RunTestsAsync("8.0", "bookworm-slim", "alpine", "noble"));
results.SelectMany(i => i).EnsureSuccess();

async Task<IEnumerable<IBuildResult>> RunTestsAsync(string framework, params string[] platforms)
{
    var publish = new DotNetPublish().WithWorkingDirectory("MySampleLib.Tests")
        .WithFramework($"net{framework}").WithConfiguration(configuration).WithNoBuild(true);
    await publish.BuildAsync(cancellationToken: cts.Token).EnsureSuccess();
    var publishPath = Path.Combine(publish.WorkingDirectory, "bin", configuration, $"net{framework}", "publish");

    var test = new VSTest().WithTestFileNames("*.Tests.dll");
    var testInDocker = new DockerRun().WithCommandLine(test).WithAutoRemove(true).WithQuiet(true)
        .WithVolumes((Path.GetFullPath(publishPath), "/app")).WithContainerWorkingDirectory("/app");
    var tasks = from platform in platforms
                let image = $"mcr.microsoft.com/dotnet/sdk:{framework}-{platform}"
                select testInDocker.WithImage(image).BuildAsync(CancellationOnFirstFailedTest, cts.Token);
    return await Task.WhenAll(tasks);
}
```

Сборка

```
async Task<IEnumerable<IBuildResult>> RunTestsAsync(string framework, params string[] platforms)
{
    var publish = new DotNetPublish()
        .WithWorkingDirectory("MySampleLib.Tests")
        .WithFramework($"net{framework}").WithConfiguration(configuration)
        .WithNoBuild(true);

    await publish.BuildAsync(cancellationToken: cts.Token).EnsureSuccess();

    var publishPath = Path.Combine(
        publish.WorkingDirectory,
        "bin",
        configuration,
        $"net{framework}",
        "publish");
```

Сборка

```
    var test = new VSTest().WithTestFileNames("*.Tests.dll");

    var testInDocker = new DockerRun()
        .WithCommandLine(test).WithAutoRemove(true).WithQuiet(true)
        .WithVolumes((Path.GetFullPath(publishPath), "/app"))
        .WithContainerWorkingDirectory("/app");

    var tasks =
        from platform in platforms
        let image = $"mcr.microsoft.com/dotnet/sdk:{framework}-{platform}"
        select testInDocker
            .WithImage(image)
            .BuildAsync(CancellationOnFirstFailedTest, cts.Token);

    return await Task.WhenAll(tasks);
}
```

Сборка

```
var results = await Task.WhenAll(
    RunTestsAsync("7.0", "bookworm-slim", "alpine"),
    RunTestsAsync("8.0", "bookworm-slim", "alpine", "noble"));
results.SelectMany(i => i).EnsureSuccess();
```

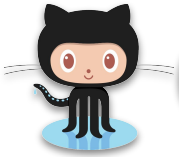
Обобщенная статистика сборки

```
Summary:
  26216 "dotnet build" finished (in 3883 ms) with exit code 0.
  06088 "dotnet publish" finished (in 4550 ms) with exit code 0.
  03004 "dotnet publish" finished (in 5281 ms) with exit code 0.
  14008 "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-noble" canceled (in 9106 ms).
  20396 "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:7.0-bookworm-slim" canceled (in 9849 ms).
  21868 "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-alpine" canceled (in 9139 ms).
  26412 "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-bookworm-slim" canceled (in 9173 ms).
  18572 "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:7.0-alpine" canceled (in 9809 ms).
  Failed MySampleLib.Tests: MySampleLib.Tests.CalculatorTests.ShouldSub(op1: 3, op2: -2, expected: 5)
    5 times
    Assert.Equal() Failure: Values differ
    Expected: 5
    Actual:   1
  Failed MySampleLib.Tests: MySampleLib.Tests.CalculatorTests.ShouldSub(op1: 3, op2: 2, expected: 1)
    5 times
    Assert.Equal() Failure: Values differ
    Expected: 1
    Actual:   5
  D:\Projects\CI-CD\MySampleLib\Calculator.cs(5,17): warning CS0169: The field 'Calculator._state' is never used
  D:\Projects\CI-CD\MySampleLib\Calculator.cs(5,17): warning CS0169: The field 'Calculator._state' is never used
  "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:7.0-bookworm-slim" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
  "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:7.0-alpine" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
  "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-bookworm-slim" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
  "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-alpine" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
  "dotnet vstest in the docker container mcr.microsoft.com/dotnet/sdk:8.0-noble" canceled with 3 errors and 7 finished tests: 2 failed, 5 passed.
2 Warning(s)
5 Error(s)
Time Elapsed 0:00:19,1881469
Running FAILED.
```

C# interactive

- 3 совместимых режима работы
  - Интерактивный REPL
  - Выполнение скриптов .csx
  - .NET проект
- 3 NuGet пакета
  - Инструмент для REPL и запуска скриптов dotnet-csi
  - Шаблон проекта сборки CSharpInteractive.Templates
  - Пакет для .NET проектов CSharpInteractive
- C# последней версии
- Отладка в IDE
- Нет модели (Task, Target, DependsOn …)
  - Нет ограничений
  - Не требует изучения
  - Можно использовать обычные практики разработки в .NET
- “Продвинутый” API сборки

C# interactive: система автоматизации сборки для .NET



GitHub/DevTeam/csharp-interactive