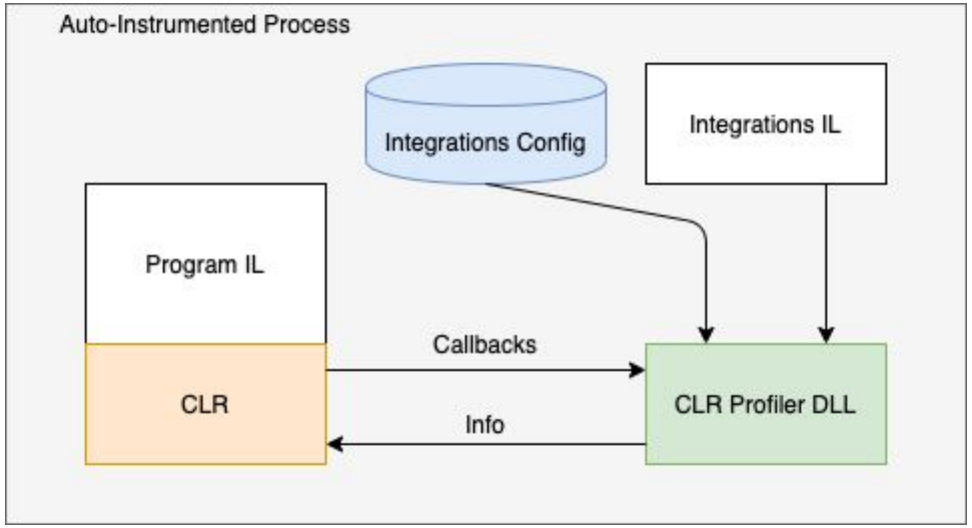
DataDog .NET Auto-Instr Architecture & Features

Paulo Janotti

High-Level Overview

- Main Components:
  a. CLR Profiler DLL (Native)
  b. Integrations DLL (Managed)
- CLR Profiler DLL
  a. Replaces calls to targeted methods w/ calls to wrappers from the Integrations DLL
- Integrations DLL
  a. Implements the wrappers, ie.: the "instrumentation", to the targeted methods
- Integrations Config
  a. Generated from method attributes on the Integrations DLL
  b. Specifies instrumentation targets and wrappers to targeted methods



Integration Configuration

- Array of Integrations
- Integration:
  - Name
  - Array of method replacements
- Method Replacement:
  - Caller (assembly)
  - Target (specific method)
  - Wrapper (method on Integrations DLL)
- Caller:
  - Target should be wrapped if called by this assembly (default: all assemblies)
- Target:
  - Method to be wrapped
- Wrapper:
  - Type and method on Integration DLL replacing direct calls to target

```
"name": "HttpMessageHandler" ,
"method_replacements" : [
  {
    "caller": {},
    "target": {
      "assembly": "System.Net.Http" ,
      "type": "System.Net.Http.HttpMessageHandler" ,
      "method": "SendAsync",
      "signature_types" : [
        "System. … `1<System.Net.Http.HttpResponseMessage>"  ,
        "System.Net.Http.HttpRequestMessage" ,
        "System.Threading.CancellationToken"
      ],
      "minimum_major" : 4,
      "minimum_minor" : 0,
      "minimum_patch" : 0,
      "maximum_major" : 4,
      "maximum_minor" : 65535,
      "maximum_patch" : 65535
    },
    "wrapper": {
      "assembly": "Datadog.… Managed, Version=1.16.0.0,…" ,
      "type": "Datadog. … .HttpMessageHandlerIntegration" ,
      "method": "HttpMessageHandler_SendAsync" ,
      "signature": "00 06 1C 1C 1C 1C 08 08 0A" ,
      "action": "ReplaceTargetMethod"
    }
  },
```

Method Wrapping Overview

- On CorProfiler::ModuleLoadFinished callback
  - Map any applicable replacement (target and wrapper) having the module id as key
- On CorProfiler::JITCompilationStarted
  - For any replacement mapped for the module id (ie.: the caller):
    - Replace all calls to the target with a call to the wrapper
- The Wrapper (aka Integration) is:
  - Written in managed code
  - Decorated with attributes to generate Integrations config file
  - Only references primitive types and instrumentation
  - Signature has objects for parameters of target plus 3 extra parameters set by the profiler
  - In charge of calling the instrumentation
  - Uses reflection to extract data (attributes, logs, etc) for instrumentation
  - In charge of calling target (with support of some helper code)

Managed Integration Example

- Target:

```
public abstract class HttpMessageHandler : IDisposable
{
   protected internal abstract Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
       CancellationToken cancellationToken);
```

- Wrapper Signature:

```
[InterceptMethod(
   TargetAssembly = SystemNetHttp,
   TargetType = HttpMessageHandler,
   TargetMethod = SendAsync,
   TargetSignatureTypes = new[] { ClrNames.HttpResponseMessageTask, ClrNames.HttpRequestMessage,
ClrNames.CancellationToken },
   TargetMinimumVersion = Major4,
   TargetMaximumVersion = Major4)]
public static object HttpMessageHandler_SendAsync(
   object handler,
   object request,
   object cancellationTokenSource,
   int opCode,
   int mdToken,
   long moduleVersionPtr)
```

Managed Integration Example

- Helpers to call target use IL emit:

```
instrumentedMethod =
  MethodBuilder<Func<HttpMessageHandler, HttpRequestMessage, CancellationToken,
Task<HttpResponseMessage>>>
     .Start(moduleVersionPtr, mdToken, opCode, SendAsync)
     .WithConcreteType(httpMessageHandler)
     .WithParameters(request, cancellationToken)
     .WithNamespaceAndNameFilters(ClrNames.GenericTask, ClrNames.HttpRequestMessage,
ClrNames.CancellationToken)
     .Build();
```

- Use reflection to read target instance or arguments:

```
HttpResponseMessage response = await sendAsync(handler, request, cancellationToken).ConfigureAwait(false);

// this tag can only be set after the response is returned
scope?.Span.SetTag(Tags.HttpStatusCode, ((int)response.StatusCode).ToString());
```

Key Design Choices

- Replace calls to target not IL rewrite of the target
  - Allows instrumentation to take target caller into account
  - Requires inlining to be disabled
- Does not reference target assemblies
  - Easier packaging (fewer dependencies)
  - Requires use of reflection and helpers

Features Summary

- Linux and Windows compatible
- Supports .NET 4.5, .NET Standard 2.0, and above
- Zero-touch instrumentation:
  - Only CLR Profiler configuration required for existing "integrations"
- Provides out-of-box integrations with various libraries and frameworks:
  - ASP.NET (Web Forms/MVC/Web API 2)
  - ADO.NET
  - WCF Server
  - Redis
  - ElasticSearch
  - MongoDB
  - PostgreSQL
  - HttpClient/HttpMessageHandler
  - WebClient/WebRequest

Final Notes

- Actively in development (as of 04/14/2020)
  - 8 PRs merged in the last 2 weeks (all from DD personnel)
  - 14 open PRs
- Repo: https://github.com/datadog/dd-trace-dotnet
- Docs: https://docs.datadoghq.com/tracing/setup/dotnet-framework/?tab=code
- .NET Auto-Instr issue: https://github.com/open-telemetry/opentelemetry-dotnet/issues/584