Profiler Auto instrumentation

Thank you for trying out the private beta of the Elastic APM .NET Profiler auto instrumentation. Below are instructions to help configure auto instrumentation for a number of common deployment environments. Currently, the common ADO.NET providers and ASP.NET Full Framework integration have been implemented. We are seeking feedback on what integrations to target next; please fill out the Google form to let us know what is important to you.

IMPORTANT

If you are adding Profiler auto instrumentation to a project that references any of the Elastic.Apm* NuGet packages, **please update these packages to version 1.11.1 first**.

Profiler auto instrumentation applies **all integrations** defined in the table below, by default. This may conflict with existing Elastic.Apm integrations that have been configured with NuGet packages, causing multiple spans to be captured. You have two different choices for how to approach this

1. Remove the NuGet package reference and configuration and use the Profiler auto instrumentation. This may be preferable for cases such as .NET Framework SqlClient integration, where the Profiler auto instrumentation can capture the command text of queries.
2. Disable specific Profiler auto instrumentation integrations using the `ELASTIC_APM_PROFILER_EXCLUDE_INTEGRATIONS` environment variable. See Profiler environment variables for more details.

Quick start

The agent can automatically instrument .NET Framework, .NET Core, and .NET applications using the .NET CLR Profiling API. The Profiling APIs provide a way to instrument an application or dependency code without code changes.

This approach works with the following

	Operating system	
Architecture	Windows	Linux
x64	.NET Framework 4.6.1+ .NET Core 2.1+ .NET 5	.NET Core 2.1+ .NET 5

and instruments the following assemblies

Integration	Description	NuGet package version(s)	Assembly version(s)
AdoNet	Captures DB spans with `DbCommand` ADO.NET provider	part of .NET	System.Data 4.0.0 - 4.*.*
AdoNet	Captures DB spans with `DbCommand` ADO.NET provider	part of .NET	System.Data.Common 4.0.0 - 5.*.*
AspNet	Registers `ElasticApmModule` for ASP.NET Full Framework integration	part of .NET Framework	System.Web 4.0.0 - 4.*.*
MySqlCommand	Captures DB spans with `MySqlCommand` ADO.NET provider	MySql.Data 6.7.0 - 8.*.*	MySql.Data 6.7.0 - 8.*.*
NpgsqlCommand	Captures DB spans with `NpgsqlCommand` ADO.NET provider	Npgsql 4.0.0 - 5.*.*	Npgsql 4.0.0 - 5.*.*
OracleCommand	Captures DB spans with `OracleCommand` ADO.NET provider	Oracle.ManagedDataAccess 4.122.0 - 4.122.*	Oracle.ManagedDataAccess 4.122.0 - 4.122.*
OracleCommand	Captures DB spans with `OracleCommand` ADO.NET provider	Oracle.ManagedDataAccess.Core 2.0.0 - 2.*.*	Oracle.ManagedDataAccess 2.0.0 - 2.*.*
SqlCommand	Captures DB spans with `SqlCommand` ADO.NET provider	part of .NET	System.Data 4.0.0 - 4.*.*
SqlCommand	Captures DB spans with `SqlCommand` ADO.NET provider	System.Data.SqlClient 4.0.0 - 4.*.*	System.Data.SqlClient 4.0.0 - 4.*.*
SqlCommand	Captures DB spans with `SqlCommand` ADO.NET provider	Microsoft.Data.SqlClient 1.0.0 - 2.*.*	Microsoft.Data.SqlClient 1.0.0 - 2.*.*
SqliteCommand	Captures DB spans with `SqliteCommand` ADO.NET provider	Microsoft.Data.Sqlite 2.0.0 - 5.*.*	Microsoft.Data.Sqlite 2.0.0 - 5.*.*
SqliteCommand	Captures DB spans with `SQLiteCommand` ADO.NET provider	System.Data.SQLite 1.0.0 - 2.*.*	System.Data.SQLite 1.0.0 - 2.*.*

Note that NuGet package versions do not necessarily align with the version of assemblies contained therein. NuGet package explorer can be used to inspect the assembly version of assemblies within a NuGet package.

IMPORTANT

The .NET CLR Profiling API allows only one profiler to be attached to a .NET process. In light of this limitation, only one solution that uses the .NET CLR profiling API should be used by an application.

Auto instrumentation using the .NET CLR Profiling API can be used in conjunction with

- the Public API to perform manual instrumentation.
- NuGet packages that perform instrumentation using a `IDiagnosticsSubscriber` to subscribe to diagnostic events.

The version number of NuGet packages referenced by a project instrumented with a profiler must be the same as the version number of profiler zip file used.

General steps:

1. Unzip the provided zip file into a folder on the host that is hosting the application to instrument.
2. Configure the following environment variables

   .NET Framework

   ```
   set COR_ENABLE_PROFILING = "1"
   set COR_PROFILER = "{FA65FE15-F085-4681-9B20-95E04F6C03CC}"
   set COR_PROFILER_PATH = "<unzipped directory>\elastic_apm_profiler.dll" ①
   set ELASTIC_APM_PROFILER_HOME = "<unzipped directory>"
   set ELASTIC_APM_PROFILER_INTEGRATIONS = "<unzipped directory>\integrations.yml"
   ```

   ① `<unzipped directory>` is the directory to which the zip file was unzipped in step 1.

   .NET Core / .NET 5 on Windows

   ```
   set CORECLR_ENABLE_PROFILING = "1"
   set CORECLR_PROFILER = "{FA65FE15-F085-4681-9B20-95E04F6C03CC}"
   set CORECLR_PROFILER_PATH = "<unzipped directory>\elastic_apm_profiler.dll" ①
   set ELASTIC_APM_PROFILER_HOME = "<unzipped directory>"
   set ELASTIC_APM_PROFILER_INTEGRATIONS = "<unzipped directory>\integrations.yml"
   ```

   ① `<unzipped directory>` is the directory to which the zip file was unzipped in step 1.

   .NET Core / .NET 5 on Linux

   ```
   export CORECLR_ENABLE_PROFILING=1
   export CORECLR_PROFILER={FA65FE15-F085-4681-9B20-95E04F6C03CC}
   export CORECLR_PROFILER_PATH="<unzipped directory>/libelastic_apm_profiler.so" ①
   export ELASTIC_APM_PROFILER_HOME="<unzipped directory>"
   export ELASTIC_APM_PROFILER_INTEGRATIONS="<unzipped directory>/integrations.yml"
   ```

   ① `<unzipped directory>` is the directory to which the zip file was unzipped in step 1.

3. Start your application in a context where the set environment variables are visible.

With this setup, the .NET runtime loads Elastic's CLR profiler into the .NET process, which loads and instantiates the APM agent early in application startup. The profiler monitors methods of interest and injects code to instrument their execution.

Instrumenting containers and services

Using global environment variables causes the Elastic profiler auto instrumentation to be loaded for **any** .NET process started on the host. Often, the environment variables should be set only for specific services or containers. The following sections demonstrate how to configure common containers and services.

Docker containers

Environment variables can be added to a Dockerfile to configure profiler auto instrumentation, and the unzipped directory can be mounted as a volume. The example below mounts the unzipped directory to `/elastic-apm-dotnet-auto-instrumentation`

```
ENV CORECLR_ENABLE_PROFILING=1
ENV CORECLR_PROFILER={FA65FE15-F085-4681-9B20-95E04F6C03CC}
ENV CORECLR_PROFILER_PATH=/elastic-apm-dotnet-auto-
instrumentation/libelastic_apm_profiler.so
ENV ELASTIC_APM_PROFILER_HOME=/elastic-apm-dotnet-auto-instrumentation
ENV ELASTIC_APM_PROFILER_INTEGRATIONS=/elastic-apm-dotnet-auto-
instrumentation/integrations.yml

ENTRYPOINT ["dotnet", "your-application.dll"]
```

Windows services

Environment variables can be added to specific Windows services by adding an entry to the Windows registry. Using PowerShell

.NET Framework service

```
$environment = [string[]]@(
  "COR_ENABLE_PROFILING=1",
  "COR_PROFILER={FA65FE15-F085-4681-9B20-95E04F6C03CC}",
  "COR_PROFILER_PATH=<unzipped directory>\elastic_apm_profiler.dll",
  "ELASTIC_APM_PROFILER_HOME=<unzipped directory>",
  "ELASTIC_APM_PROFILER_INTEGRATIONS=<unzipped directory>\integrations.yml")

Set-ItemProperty HKLM:SYSTEM\CurrentControlSet\Services\<service-name> -Name
Environment -Value $environment
```

.NET Core service

```
$environment = [string[]]@(
  "CORECLR_ENABLE_PROFILING=1",
  "CORECLR_PROFILER={FA65FE15-F085-4681-9B20-95E04F6C03CC}",
  "CORECLR_PROFILER_PATH=<unzipped directory>\elastic_apm_profiler.dll", ①
  "ELASTIC_APM_PROFILER_HOME=<unzipped directory>",
  "ELASTIC_APM_PROFILER_INTEGRATIONS=<unzipped directory>\integrations.yml")

Set-ItemProperty HKLM:SYSTEM\CurrentControlSet\Services\<service-name> -Name
Environment -Value $environment ②
```

① `<unzipped directory>` is the directory to which the zip file was unzipped

② `<service-name>` is the name of the Windows service.

With PowerShell

```
Restart-Service <service-name>
```

Internet Information Services (IIS)

For IIS versions *before* IIS 10, it is **not** possible to set environment variables scoped to a specific application pool, so environment variables need to set globally.

For IIS 10 *onwards*, environment variables can be set for an application pool using AppCmd.exe. With PowerShell

.NET Framework

```
$appcmd = "$($env:systemroot)\system32\inetsrv\AppCmd.exe"
$appPool = "<application-pool>" ①
$environment = @{
  COR_ENABLE_PROFILING = "1"
  COR_PROFILER = "{FA65FE15-F085-4681-9B20-95E04F6C03CC}"
  COR_PROFILER_PATH = "<unzipped directory>\elastic_apm_profiler.dll" ②
  ELASTIC_APM_PROFILER_HOME = "<unzipped directory>"
  ELASTIC_APM_PROFILER_INTEGRATIONS = "<unzipped directory>\integrations.yml"
  COMPlus_LoaderOptimization = "1" ③
}

$environment.Keys | ForEach-Object {
  & $appcmd set config -section:system.applicationHost/applicationPools
/+"[name='$appPool'].environmentVariables.[name='$_',value='$($environment[$_])']"
}
```

① `<application-pool>` is the name of the Application Pool your application uses

② `<unzipped directory>` is the directory to which the zip file was unzipped

③ Forces assemblies **not** to be loaded domain-neutral. There is currently a limitation where

Profiler auto-instrumentation cannot instrument assemblies when they are loaded domain-neutral. This limitation is expected to be removed in future, but for now, can be worked around by setting this environment variable. See the Microsoft documentation for further details.

.NET Core

```
$appcmd = "$($env:systemroot)\system32\inetsrv\AppCmd.exe"
$appPool = "<application-pool>" ①
$environment = @{
  CORECLR_ENABLE_PROFILING = "1"
  CORECLR_PROFILER = "{FA65FE15-F085-4681-9B20-95E04F6C03CC}"
  CORECLR_PROFILER_PATH = "<unzipped directory>\elastic_apm_profiler.dll" ②
  ELASTIC_APM_PROFILER_HOME = "<unzipped directory>"
  ELASTIC_APM_PROFILER_INTEGRATIONS = "<unzipped directory>\integrations.yml"
}

$environment.Keys | ForEach-Object {
  & $appcmd set config -section:system.applicationHost/applicationPools
/+"[name='$appPool'].environmentVariables.[name='$_',value='$($environment[$_])']"
}
```

① `<application-pool>` is the name of the Application Pool your application uses

② `<unzipped directory>` is the directory to which the zip file was unzipped

IMPORTANT Ensure that the location of the `<unzipped directory>` is accessible and executable to the Identity account under which the Application Pool runs.

Once environment variables have been set, stop and start IIS so that applications hosted in IIS will see the new environment variables

```
net stop /y was
net start w3svc
```

systemd / systemctl

Environment variables can be added to specific services run with systemd by creating an environment.env file containing the following

```
CORECLR_ENABLE_PROFILING=1
CORECLR_PROFILER={FA65FE15-F085-4681-9B20-95E04F6C03CC}
CORECLR_PROFILER_PATH=/elastic-apm-dotnet-auto-
instrumentation/libelastic_apm_profiler.so
ELASTIC_APM_PROFILER_HOME=/elastic-apm-dotnet-auto-instrumentation
ELASTIC_APM_PROFILER_INTEGRATIONS=/elastic-apm-dotnet-auto-
instrumentation/integrations.yml
```

Then adding an `EnvironmentFile` entry to the service's configuration file that references the path to the environment.env file

```
[Service]
EnvironmentFile=/path/to/environment.env
ExecStart=<command> ①
```

① the command that starts your service

After adding the `EnvironmentFile` entry, restart the service

```
systemctl reload-or-restart <service>
```

Profiler environment variables

The profiler auto instrumentation has its own set of environment variables to manage the instrumentation

`ELASTIC_APM_PROFILER_HOME`

The home directory of the profiler auto instrumentation.

`ELASTIC_APM_PROFILER_INTEGRATIONS` ***(optional)***

The path to the integrations.yml file that determines which methods to target for auto instrumentation. If not specified, the profiler will assume an integrations.yml exists in the home directory specified by `ELASTIC_APM_PROFILER_HOME` environment variable.

`ELASTIC_APM_PROFILER_EXCLUDE_INTEGRATIONS` ***(optional)***

A semi-colon separated list of integrations to exclude from auto-instrumentation. Valid values are those defined in the `Integration` column in the integrations table above.

`ELASTIC_APM_PROFILER_EXCLUDE_PROCESSES` ***(optional)***

A semi-colon separated list of process names to exclude from auto-instrumentation. For example, `dotnet.exe;powershell.exe`. Can be used in scenarios where profiler environment variables have a global scope that would end up auto-instrumenting applications that should not be.

`ELASTIC_APM_PROFILER_EXCLUDE_SERVICE_NAMES` ***(optional)***

A semi-colon separated list of APM service names to exclude from auto-instrumentation. Values defined are checked against the value of `ELASTIC_APM_SERVICE_NAME` environment variable.

`ELASTIC_APM_PROFILER_LOG` ***(optional)***

The log level at which the profiler should log. Valid values are

- trace
- debug
- info
- warn

- error
- none

The default value is `warn`. More verbose log levels like `trace` and `debug` can affect the runtime performance of profiler auto instrumentation, so are recommended *only* for diagnostics purposes.

`ELASTIC_APM_PROFILER_LOG_DIR` ***(optional)***

The directory in which to write profiler log files. If unset, defaults to

- `%PROGRAMDATA%\elastic\apm-agent-dotnet\logs` on Windows
- `/var/log/elastic/apm-agent-dotnet` on Linux

If the default directory cannot be written to for some reason, the profiler will try to write log files to a `logs` directory in the home directory specified by `ELASTIC_APM_PROFILER_HOME` environment variable.