EventStoreDB .NET Environment Installation

Overview

Welcome to the .NET (C#) example of Event Store's **From Scratch** series. This series lets you quickly overcome the common challenges of setting up and configuring a new development environment and focus on advancing your EventStoreDB skills.

The **From Scratch** series provides working code examples for basic reads and writes to EventStoreDB, a tested environment to run the code, and instructions that clearly describe the steps required to run the code successfully.

Each **From Scratch** repository provides the following:

- A working Github Codespaces environment
- Instructions on running EventStoreDB locally
- Instructions to set up a similar project on your own

We recommend you progress through the **From Scratch** projects in the following order:

1. Run the code in Codespaces
2. Clone the For Scratch GitHub repo, and follow the instructions to run it locally
3. Build your own project

This document provides detailed instructions for **setting up your own .NET project in EventStoreDB.** ***This is the recommended third stage in Event Store's From Scratch .NET series.***

Other clients in the **From Scratch** series include:

- Node
- Java
- Python

Topics covered

1. Setting up a .NET project
2. Adding EventStoreDB dependencies to the .NET project
3. Update the .NET Program.cs file to read from and write to EventStoreDB
4. Start a local Docker container running EventStoreDB
5. Configure .NET for GitHub and create a GitHub repository

This is intended as a baseline working example of an EventStoreDB .NET project. Your .NET projects may be significantly more complex.

1. Setting up a .NET project

Installing .NET

Similar to Java, .NET has a runtime environment to run compiled code. There is also an SDK, or software development kit, which includes a compiler to write and compile code.

If you are on a Mac, run the following command to access the SDK. If prompted, provide your system password.

```
brew install --cask .NET-sdk.
```

There are alternate methods for installing a .NET SDK. If you need additional support, please refer to the .NET documentation.

After installing the .NET SDK, run the following command.

```
dotnet --version
```

It should return the version of .NET you are running.

```
dotnet --version
8.0.203
```

The steps described in this document are very similar to those provided by Microsoft at the following location:

- https://learn.microsoft.com/en-us/dotnet/core/tools/global-tools-how-to-create#create-a-project

Setting up a project

Now that you have a .NET SDK, you can set up your own project.

Useful documentation:

- *https://learn.microsoft.com/en-us/.NET/core/tools/global-tools-how-to-create#create-a-project*
- *https://learn.microsoft.com/en-us/.NET/core/tools/.NET-new*

Create a new directory named dotnetFromScratch, or use a name you can remember.

```
mkdir dotnetFromScratch
```

Navigate to your new directory.

```
cd dotnetFromScratch
```

Build a project directory structure

.NET provides a template service. Run the following command to create the directory **EventStoreDB.demo** using .NET 8.0 (which should match the .NET SDK installed on your machine).

```
dotnet new console -n EventStoreDB.demo -f net8.0
```

The **EventStoreDB.demo** directory structure is as follows:

```
EventStoreDB.demo
├── Program.cs
├── bin
│   └── Debug
│       └── net8.0
│           ├── EventStoreDB.demo
│           ├── EventStoreDB.demo.deps.json
│           ├── EventStoreDB.demo.dll
│           ├── EventStoreDB.demo.pdb
│           └── EventStoreDB.demo.runtimeconfig.json
├── EventStoreDB.demo.csproj
└── obj
    ├── Debug
    │   └── net8.0
    │       ├── apphost
    │       ├── EventStoreDB.demo.AssemblyInfo.cs
    │       ├── EventStoreDB.demo.AssemblyInfoInputs.cache
    │       ├──
EventStoreDB.demo.GeneratedMSBuildEditorConfig.editorconfig
    │       ├── EventStoreDB.demo.GlobalUsings.g.cs
    │       ├── EventStoreDB.demo.assets.cache
    │       ├── EventStoreDB.demo.csproj.CoreCompileInputs.cache
    │       ├── EventStoreDB.demo.csproj.FileListAbsolute.txt
    │       ├── EventStoreDB.demo.dll
    │       ├── EventStoreDB.demo.genruntimeconfig.cache
    │       ├── EventStoreDB.demo.pdb
    │       ├── ref
    │       │   └── EventStoreDB.demo.dll
    │       └── refint
    │           └── EventStoreDB.demo.dll
    ├── EventStoreDB.demo.csproj.nuget.dgspec.json
```

```
├── EventStoreDB.demo.csproj.nuget.g.props
├── EventStoreDB.demo.csproj.nuget.g.targets
├── project.assets.json
└── project.nuget.cache
```

Verify It Works

.NET projects built with `dotnet new` include a program.cs file that prints "*Hello, World!*" to your console.

Navigate to the EventStoreDB.demo folder and run the following command to verify your environment is functioning properly.

```
dotnet run
```

This command executes the directory's Program.cs file and should produce the following in the console.

```
Hello, World!
```

Congratulations! You have successfully created a .NET project.

2. Adding EventStoreDB client dependencies to the .Net project

Note: Similar instructions are included in the Event Store Developer Center: https://developers.eventstore.com/clients/grpc/#connecting-to-EventStoreDB

Use the following `dotnet add` command to add the EventStoreDB client dependencies to your project.

```
dotnet add package EventStore.Client.Grpc.Streams --version 23.1.0
```

3. Update the .NET Program.cs file to read from and write to EventStoreDB

Program.cs is the main entry point for executing .NET code. For this sample example, you will modify the main Program.cs to read and write to EventStoreDB.

Using the text editor of your choice, or an IDE, replace the prebuilt Program.cs with the following content.

```
using System.Text;
using EventStore.Client;

Console.WriteLine("Hello, World!");

var settings =
EventStoreClientSettings.Create("esdb://localhost:2113?tls=false");
await using var client = new EventStoreClient(settings);
await AppendToStream(client);
await ReadFromStream(client);

return;

static async Task AppendToStream(EventStoreClient client) {
     #region append-to-stream

     var eventData = new EventData(
          Uuid.NewUuid(),
          "some-event",
          "{\"id\": \"1\" \"value\": \"some value\"}"u8.ToArray()
     );

     await client.AppendToStreamAsync(
          "SampleContent",
          StreamState.Any,
          new List<EventData> {
               eventData
          }
     );

     #endregion append-to-stream
}

static async Task ReadFromStream(EventStoreClient client) {
     #region read-from-stream

     var events = client.ReadStreamAsync(
          Direction.Forwards,
          "SampleContent",
```

```
            StreamPosition.Start
        );

        #endregion read-from-stream

        #region iterate-stream

        await foreach (var @event in events)
Console.WriteLine(Encoding.UTF8.GetString(@event.Event.Data.ToArray(
)));

        #endregion iterate-stream

        #region #read-from-stream-positions

        Console.WriteLine(events.FirstStreamPosition);
        Console.WriteLine(events.LastStreamPosition);

        #endregion
}

static async Task ReadFromStreamMessages(EventStoreClient client) {
        #region read-from-stream-messages

        var streamPosition = StreamPosition.Start;
        var results = client.ReadStreamAsync(
            Direction.Forwards,
            "some-stream",
            streamPosition
        );

        #endregion read-from-stream-messages

        #region iterate-stream-messages

        await foreach (var message in results.Messages)
            switch (message) {
                case StreamMessage.Ok ok:
                    Console.WriteLine("Stream found.");
                    break;
```

```
                case StreamMessage.NotFound:
                     Console.WriteLine("Stream not found.");
                     return;

                case StreamMessage.Event(var resolvedEvent):

Console.WriteLine(Encoding.UTF8.GetString(resolvedEvent.Event.Data.S
pan));
                     break;

                case StreamMessage.FirstStreamPosition(var sp):
                     Console.WriteLine($"{sp} is after
{streamPosition}; updating checkpoint.");
                     streamPosition = sp;
                     break;

                case StreamMessage.LastStreamPosition(var sp):
                     Console.WriteLine($"The end of the stream is
{sp}");
                     break;

                default:
                     break;
           }

     #endregion iterate-stream-messages
}
```

4. Start a local Docker container running EventStoreDB

If you have yet to do so, install Docker: https://docs.docker.com/engine/install/.

Run the following command to initiate an unsecured single instance EventStoreDB cluster locally.

```
docker run -d --name esdb-node -it -p 2113:2113 -p 1113:1113 \
     eventstore/eventstore:lts --insecure --run-projections=All \
     --enable-external-tcp --enable-atom-pub-over-http
```

Use the following command to run .NET.

```
dotnet run
```

Like earlier examples in the From Scratch series, you can view the EventStore WebUI by pointing your browser to http://localhost:2113. Select the "Stream Browser" tab to view the list of streams. After running `dotnet run`, you should see your newly created stream and event.

5. Configure .NET for GitHub and create a GitHub repository

Programming languages that are compiled have "source" files. When the project is built, "compiled" or "build" files are generated. It is best practice to store only your source code in GitHub.

Since your project directory contains build artifacts generated when `dotnet run` is executed, it is important to write a .gitignore file to specify which content should **not** be managed by git.

.NET has a built-in command to generate a .gitignore file.

Run this command in the EventStoreDB.demo directory to generate a .gitignore file.

```
dotnet new gitignore
```

To initialize the directory as a local git repo run the following command.

```
git init
```

Create a repo on GitHub and push your directory as the first commit

Navigate to your GitHub repositories, and create a new repo titled FromScratch_dotnet (or a name of your choosing) by selecting the green "New" button in the upper right of the screen.

After creating the repo, GitHub presents a page with instructions. Follow the instructions titled "**...or push an existing repository from the command line**." You will see two steps below that are not included in GitHub. Ensure you follow the steps below.

Run the following commands to link and push your local Git repo to your GitHub repo. Remember to replace `<YOUR REPO NAME>` with the name of your repository.

```
git remote add https://github.com/<YOUR REPO NAME>.git
```

```
git branch -M main
```

```
git add -A
```

```
git commit -m "first commit"
```

```
git push -u origin main
```

Congratulations! You now have a working .NET project that includes a basic write and read using EventStoreDB. This can be the foundation for building more advanced and complete projects.

Next Steps

Now that you have successfully created a .NET project, you have completed the From Scratch .NET lessons. Please feel free to venture into another **From Scratch** series. Event Store offers similar content for Python, Java, and Node.js.

Or continue your learning, you can find additional examples in the following repo:

https://github.com/EventStore/samples

In particular, we recommend the Quickstart examples here:

https://github.com/EventStore/samples/tree/main/Quickstart