Integrating Stella Now SDK with local NanoMQ node

(C# EXAMPLE)

1. Overview

This guide walks through the steps to integrate StellaNow SDK with a local instance of NanoMQ using a basic setup with no authentication. NanoMQ is an MQTT broker optimized for lightweight, high-performance messaging.

2. Prerequisites

Before proceeding, ensure you have the following installed:

- NanoMQ (latest version)
- .NET 8.0+ (for StellaNow SDK)
- StellaNow SDK
    - Github: https://github.com/stella-systems/stellanow-sdk-csharp
    - NuGet: https://www.nuget.org/packages/StellaNowSDK/0.4.0-rc1
- StellaNow CLI - https://pypi.org/project/stellanow-cli/
- MQTT Explorer (optional, for debugging MQTT messages)

3. Example Implementation

For a full working example of integration, you can check out the Stella Now SDK C# repository. Inside you can find a demo app.
Direct Link - https://github.com/stella-systems/stellanow-sdk-csharp/blob/develop/StellaNowSDKDemo/src/Program.cs

4. Installing and Running NanoMQ Locally

Run following command to download and run the most basic NanoMQ node:

```
docker run -d --name nanomq -p 1883:1883 -p 8083:8083 -p 8883:8883 emqx/nanomq:latest
```

5.Configuring StellaNow SDK for NanoMQ

StellaNow SDK needs to be configured to connect to NanoMQ without authentication.

Step 1: Add StellaNow SDK to your project

Add StellaNow SDK to your project via NuGet:

```
dotnet add package StellaNowSDK --version 0.4.0
```

Step 2: (Optional) Create environment config class

StellaNow SDK needs to know where to connect. This is needed during registration of the SDK instance and looks as follows:

```
public class StellaNowLocalEnvironmentConfig : StellaNowEnvironmentConfig
{
    protected override string ApiBaseUrl => "https://api.dev.stella.cloud";
    public override string BrokerUrl => "mqtt://localhost:1883";
}
```

Breakdown:

- ApiBaseUrl – URL base for where the API of StellaNow Platform is hosted
- BrokerUrl – URL with protocol and port of the MQTT broker

Step 2: Register StellaNow SDK for No-Auth Connection (e.g. Local MQTT Broker – NanoMQ)

The function AddStellaNowSdkWithMqttAndNoAuth is a helper extension method that registers the necessary dependencies into an IServiceCollection to set up the StellaNow SDK with MQTT connectivity using no authentication.

```
var services = new ServiceCollection();

services.AddStellaNowSdkWithMqttAndNoAuth(
    new StellaNowLocalEnvironmentConfig(),
    new StellaNowConfig(
        Environment.GetEnvironmentVariable("ORGANIZATION_ID")!,
        Environment.GetEnvironmentVariable("PROJECT_ID")!
    )
);
```

Breakdown:

1. Creates a new ServiceCollection instance – This acts as the DI (Dependency Injection) container.
2. Registers the StellaNow SDK with MQTT (No Authentication)
   - Environment Configuration → StellaNowLocalEnvironmentConfig()
     - Specifies that the SDK should connect to a local MQTT broker (like NanoMQ).
   - StellaNow Configuration → StellaNowConfig(...)
     - Organization & Project IDs are fetched from environment variables.

How to Retrieve the SDK Instance?

After configuring the DI container, resolve the IStellaNowSdk instance as follows:

```
var serviceProvider = services.BuildServiceProvider();
var stellaSdk = serviceProvider.GetRequiredService<IStellaNowSdk>();
```

6.Generating Event Classes with the StellaNow CLI

This section covers how to install and configure the StellaNow CLI for code generation, list available events, plan changes, and finally generate the event classes for use in your .NET project.

6.1. Installing the CLI

Python & pip: Ensure you have Python 3.x and pip installed on your system.

```
pip install stellanow-cli
```

This will install the stellanow command globally.

Verify Installation:

```
stellanow --help
```

You should see the usage information for the StellaNow CLI.

6.2. Configuring the CLI

Before you can generate event classes, configure the code-generator-service. This tells the CLI which StellaNow API endpoint and credentials to use:

```
stellanow --profile MY_PROFILE configure code-generator-service
```

- **Base URL** for the StellaNow API
- **Username and Password** for your StellaNow account
- **Organization ID** (UUID)

The configuration is saved in ~/.stellanow/config.ini. You can repeat this step for different profiles (e.g., dev, test, prod).

6.3. Listing Available Events

You can list available events (defined in your StellaNow project) using:

```
stellanow code-generator-service events --project_id YOUR_PROJECT_ID
```

This displays a table with each event's ID, name, status, and timestamps. Note which events you want to generate.

6.4. Planning Changes

If you already have previously generated classes in your project, you can compare them against the latest event definitions using the plan command:

```
stellanow code-generator-service plan --project_id YOUR_PROJECT_ID --input_dir .
```

- **--input_dir** is the folder containing any previously generated classes.
- The plan command will show a summary of what has changed in the event definitions since your last generation (e.g., new fields, removed fields, etc.).

6.5. Generating Event Classes

To actually generate or update the C# event classes, run:

```
stellanow --profile <profile-id> code-generator-service events -project_id <project-uuid>
```

Key Flags Explained:

- **--project_id:** Specifies which StellaNow project's events to generate.
- **--namespace (-n):** The C# namespace root. Event classes will go under MyApp.Events, and model classes will go under MyApp.Models.
- **--destination (-d):** Relative path to where to output the generated .cs files.
- **--language (-l):** Programming language, typically csharp for .NET.
- **--events (-e):** Comma-separated list of specific event names or IDs to generate. If omitted, the CLI will generate all available events.
- **--force (-f):** Overwrites existing files if there are changes (otherwise, you might be prompted to confirm overwrites).

When the command completes, you'll have a new or updated set of C# classes corresponding to your StellaNow events and models.

6.6. Example of a Generated Class (with MIT License Header)

Below is a sample of what a generated C# event class might look like. The StellaNow CLI can optionally include a MIT License header at the top of each file to ensure compliance:

```
/*
 *  This file is auto-generated by StellaNowCLI. DO NOT EDIT.
 *
 *  Event ID: 4eaa06cc-9bc2-4f0f-8e16-94d5e5fa7bed
 *  Generated: 2024-11-05T15:10:25Z
 */

using StellaNowSDK.Messages;

namespace StellaNowSDKDemo.Messages;

public record UserLoginMessage(
    [property: Newtonsoft.Json.JsonIgnore] string patronId,
    [property: Newtonsoft.Json.JsonProperty("user_id")] string UserId,
    [property: Newtonsoft.Json.JsonProperty("timestamp")] DateTime Timestamp
    ) : StellaNowMessageBase("user_login", new List<EntityType>{ new EntityType("patron",
patronId) });
```

Key Points:

- **Auto-Generated Note:** Indicates this file should not be manually edited.

- **Constructor Fields:** Mapped from the StellaNow event's schema (user_id, timestamp, etc.).
- **Entity References:** patronId is passed to EntityType("patron", patronId) to link the event to a particular entity in StellaNow.

6.7. Integrating Generated Classes with the StellaNow SDK

After generating classes, copy or reference them in your .NET project. Below is how they can be used alongside the StellaNowSdk:

```
using Microsoft.Extensions.Logging;
using StellaNowSDK;
using MyApp.Events; // where UserLoginMessage lives

public class UserLoginPublisher
{
    private readonly IStellaNowSdk _sdka;
    private readonly ILogger<UserLoginPublisher> _logger;

    public UserLoginPublisher(IStellaNowSdk sdk, ILogger<UserLoginPublisher> logger)
    {
        _sdk = sdk;
        _logger = logger;
    }

    public void PublishUserLogin(string patronId, string userId)
    {
        var loginEvent = new UserLoginMessage(
            patronId: patronId,
            UserId: userId,
            Timestamp: DateTime.UtcNow
        );

        // Queue for dispatch
        _sdk.SendMessage(loginEvent, OnMessageSent);
    }

    private void OnMessageSent(StellaNowEventWrapper wrapper)
    {
        _logger.LogInformation(
            "UserLoginMessage sent: {MessageId}", wrapper.Value.Metadata.MessageId
        );
    }
}
```

1. **Instantiation:** Create a new UserLoginMessage, passing the required constructor parameters (mapped to your event schema).
2. **Dispatch:** Call _sdk.SendMessage(...), which enqueues and eventually publishes the message to your MQTT broker (e.g., NanoMQ).
3. **Callbacks:** Optionally provide a callback (OnMessageSent) to confirm the message was processed.

6.8. Best Practices and Tips

- **Version Control:** Check the generated classes into source control. Keep track of which CLI version was used to generate them.
- **Regenerate on Schema Changes:** Anytime event definitions change in StellaNow, run plan to see differences, then generate to update your code.
- **Use Profiles:** If you work across multiple environments (e.g., dev/test/prod), set up separate CLI profiles to quickly switch configurations.
- **Avoid Manual Edits:** If you need extra functionality, consider partial classes or separate extension classes instead of editing generated files directly. This way, your changes won't be overwritten by future regenerations.

7. Running the Application and Observing Results

After completing the setup for NanoMQ, configuring the StellaNow SDK, and generating your event classes, you're ready to run your new application and verify that messages are being published to the local NanoMQ instance.

7.1. Starting the Local NanoMQ Broker

If you haven't already, start or verify that NanoMQ is running locally. For example:

```
docker run -it --rm -p 1883:1883 emqx/nanomq
```

Example command to start NanoMQ (exact command may vary by OS/version)

This command:

- Pulls the NanoMQ image (emqx/nanomq) if it isn't already on your system.
- Exposes port 1883 for MQTT traffic on your local machine.

- Runs NanoMQ in the foreground so you can view logs in real time.

Ensure NanoMQ is listening on the same host/port that you configured in your StellaNowLocalEnvironmentConfig (default:mqtt://localhost:1883).

7.2. Running Your .NET Application

From your project directory, run:

```
dotnet run
```

Depending on your code, you may pass in optional arguments (e.g., a message count). For instance:

```
dotnet run -- 5
```

This could tell your application to publish 5 messages before exiting. If no argument is provided, it might publish indefinitely or until you press ENTER.

Sample Console Logs

Below is an example of what you might see in the console if you have logging enabled:

```
[2025-02-28 01:59:23] info: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      SDK Client ID is "StellaNowSDK_1ieQVbCNYN"
[2025-02-28 01:59:23] info: StellaNowSDK.Services.StellaNowSdk[0]
      Starting
[2025-02-28 01:59:23] info: StellaNowSDKDemo.Program[0]
      Press 'ENTER' to stop the application!
[2025-02-28 01:59:23] info: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      Started Reconnection Monitor
[2025-02-28 01:59:23] dbug: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Start Processing Message Queue
[2025-02-28 01:59:23] info: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      Connecting
[2025-02-28 01:59:23] info: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      Connected
[2025-02-28 01:59:29] dbug: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Queueing Message: a4822924-4ada-4de9-be82-36376f3ebff1
[2025-02-28 01:59:29] info: StellaNowSDKDemo.Program[0]
      New Message Queued
[2025-02-28 01:59:29] info: StellaNowSDKDemo.Program[0]
      Queue has messages True and count is 1
[2025-02-28 01:59:29] info: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Attempting to send message: a4822924-4ada-4de9-be82-36376f3ebff1
[2025-02-28 01:59:29] dbug: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      Sending Message: a4822924-4ada-4de9-be82-36376f3ebff1
[2025-02-28 01:59:29] info: StellaNowSDKDemo.Program[0]
      Send Confirmation: a4822924-4ada-4de9-be82-36376f3ebff1
[2025-02-28 01:59:30] dbug: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Queueing Message: e6e941ea-f250-4d6a-9f10-cba4ee4fb742
[2025-02-28 01:59:30] info: StellaNowSDKDemo.Program[0]
      New Message Queued
[2025-02-28 01:59:30] info: StellaNowSDKDemo.Program[0]
      Queue has messages True and count is 1
[2025-02-28 01:59:30] info: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Attempting to send message: e6e941ea-f250-4d6a-9f10-cba4ee4fb742
[2025-02-28 01:59:30] dbug: StellaNowSDK.Sinks.Mqtt.StellaNowMqttSink[0]
      Sending Message: e6e941ea-f250-4d6a-9f10-cba4ee4fb742
[2025-02-28 01:59:30] info: StellaNowSDKDemo.Program[0]
      Send Confirmation: e6e941ea-f250-4d6a-9f10-cba4ee4fb742
[2025-02-28 01:59:31] dbug: StellaNowSDK.Services.StellaNowMessageQueue[0]
      Queueing Message: b6990887-a45d-4e8b-96f8-bb438ac1d4a8
[2025-02-28 01:59:31] info: StellaNowSDKDemo.Program[0]
```

What's happening:

- **SDK Startup:** The StellaNow SDK starts, attempts to connect to NanoMQ, and logs a successful connection.
- **Message Publishing:** Your custom event classes (e.g., UserLoginMessage) are queued and eventually dispatched to the broker.
- Send Confirmation: A callback indicates the message was successfully published to the MQTT broker.

- Shutdown: When you press ENTER or hit your desired message count, the SDK gracefully stops, ensuring all messages are sent.

7.3. Viewing Messages on NanoMQ

To confirm that messages are actually being published, you can subscribe to all topics (or specific topics) on your NanoMQ broker using one of the following:

MQTT Explorer (GUI):

- Download and install MQTT Explorer.
- Connect to mqtt://localhost:1883.
- Observe the incoming messages under the relevant topics (e.g., in/YOUR_ORGANIZATION_ID, if that's how you configured your StellaNow SDK).