Adding monitoring

Start

Let’s update our Trading microservice so it can start collecting, aggregating and exporting metrics related to the purchase process.

In Trading repo

1. Add and inject purchase counters to PurchaseStateMachine:

```
public class PurchaseStateMachine : MassTransitStateMachine<PurchaseState>
{
  …
  private readonly ServiceSettings settings;
  private readonly Counter<int> purchaseStartedCounter;
  private readonly Counter<int> purchaseSuccessCounter;
  private readonly Counter<int> purchaseFailedCounter;
  …

  public PurchaseStateMachine(
    MessageHub hub,
    ILogger<PurchaseStateMachine> logger,
    IConfiguration configuration)
  {
    …
    this.logger = logger;

    settings = configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
    Meter meter = new(settings.ServiceName);
    purchaseStartedCounter = meter.CreateCounter<int>("PurchaseStarted");
    purchaseSuccessCounter = meter.CreateCounter<int>("PurchaseSuccess");
    purchaseFailedCounter = meter.CreateCounter<int>("PurchaseFailed");
  }
  …
}
```

2. Count start, success and failed events:

```
public class PurchaseStateMachine : MassTransitStateMachine<PurchaseState>
{
  …
  private void ConfigureInitialState()
  {
    Initially(
      When(PurchaseRequested)
```

```
        .Then(context =>
        {
           …
          logger.LogInformation(…);
          purchaseStartedCounter.Add(1, new KeyValuePair<string, object>(nameof(context.Saga.ItemId),
context.Saga.ItemId));
        })
        …
        .Catch<Exception>(ex => ex.
          Then(context =>
          {
            …
            logger.LogError(…);
            purchaseFailedCounter.Add(1, new KeyValuePair<string, object>(nameof(context.Saga.ItemId),
context.Saga.ItemId));
          })
          …
    );
  }

  private void ConfigureAccepted()
  {
    During(Accepted,
      …
      When(GrantItemsFaulted)
        .Then(context =>
        {
          …
          logger.LogError(…);
          purchaseFailedCounter.Add(1, new KeyValuePair<string, object>(nameof(context.Saga.ItemId),
context.Saga.ItemId));
        })
        …
      );
  }

  private void ConfigureItemsGranted()
  {
    During(ItemsGranted,
      …
      When(GilDebited)
        .Then(context =>
        {
          …
          logger.LogInformation(…);
          purchaseSuccessCounter.Add(1, new KeyValuePair<string, object>(nameof(context.Saga.ItemId),
context.Saga.ItemId));
        })
        …
```

```
            When(DebitGilFaulted)
                ...
                .Then(context =>
                {
                    ...
                    logger.LogInformation(…);
                    purchaseFailedCounter.Add(1, new KeyValuePair<string, object>(nameof(context.Saga.ItemId),
context.Saga.ItemId));
                })
                ...
        );
    }
    ...
}
```

3. Add the Prometheus exporter NuGet package:

```
dotnet add package OpenTelemetry.Exporter.Prometheus --version 1.2.0-rc5
```

4. Register metrics services on Startup:

```
public class Startup
{
    ...
    public void ConfigureServices(IServiceCollection services)
    {
        ...
        services.AddSeqLogging(Configuration)
                .AddTracing(Configuration);

        services.AddOpenTelemetryMetrics(builder =>
        {
            var serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
            builder.AddMeter(serviceSettings.ServiceName)
                    .AddMeter("MassTransit")
                    .AddHttpClientInstrumentation()
                    .AddAspNetCoreInstrumentation()
                    .AddPrometheusExporter();
        });
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            ...
        }
        app.UseOpenTelemetryPrometheusScrapingEndpoint();

        app.UseHttpsRedirection();
        …
    }
    …
}
```

5. Run all microservices

6. Perform a purchase or two from the Frontend

7. Wait a few seconds

8. Browse to the metrics endpoint at:

   http://localhost:5006/metrics

9. Notice the exported metrics

10. Commit and push changes.

In the next lesson you will learn how to stand up a Prometheus server via Docker.