Generalizing the monitoring configuration

Start

Let's now extract the Prometheus configuration we currently have in our Trading microservice into our Common library so that enabling metrics becomes very straightforward for future microservices.

In Common repo

1. Copy the Prometheus exporter package reference from Trading project to Play.Common.csproj:

```
<Project Sdk="Microsoft.NET.Sdk">
  …
  <ItemGroup>
    …
    <PackageReference Include="OpenTelemetry.Exporter.Jaeger" Version="1.3.0-beta.1" />
    <PackageReference Include="OpenTelemetry.Exporter.Prometheus" Version="1.2.0-rc5" />
    <PackageReference Include="OpenTelemetry.Extensions.Hosting" Version="1.0.0-rc9.3" />
    …
  </ItemGroup>
</Project>
```

2. Add the AddMetrics method to OpenTelemetry\Extensions.cs:

```
public static IServiceCollection AddMetrics(this IServiceCollection services, IConfiguration config)
{
    services.AddOpenTelemetryMetrics(builder =>
    {
        var serviceSettings = config.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
        builder.AddMeter(serviceSettings.ServiceName)
               .AddMeter("MassTransit")
               .AddHttpClientInstrumentation()
               .AddAspNetCoreInstrumentation()
               .AddPrometheusExporter();
    });

    return services;
}
```

3. Add a UseInstrumentation() calls to MassTransit\Extensions.cs:

```
public static class Extensions
{
    …
```

```
    public static void UsingPlayEconomyRabbitMq(
        this IBusRegistrationConfigurator configure,
        Action<IRetryConfigurator> configureRetries = null)
    {
            …
            configurator.UseMessageRetry(configureRetries);
            configurator.UseInstrumentation(serviceName: serviceSettings.ServiceName);
        });
    }

    public static void UsingPlayEconomyAzureServiceBus(
        this IBusRegistrationConfigurator configure,
        Action<IRetryConfigurator> configureRetries = null)
    {
        configure.UsingAzureServiceBus((context, configurator) =>
        {
            …
            configurator.UseMessageRetry(configureRetries);
            configurator.UseInstrumentation(serviceName: serviceSettings.ServiceName);
        });
    }
}
```

4. Commit and push changes

5. Wait for the GitHub workflow to complete

In Trading repo

6. Bump the version of Play.Common to the just published NuGet package version

7. Remove OpenTelemetry package references

8. Use the new AddMetrics method in Startup.ConfigureServices and remove call to AddOpenTelemetryMetrics:

```
public void ConfigureServices(IServiceCollection services)
{
    …
    services.AddSeqLogging(Configuration)
            .AddTracing(Configuration)
            .AddMetrics(Configuration);

    services.AddOpenTelemetryMetrics(builder =>
    {
```

```
    var serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
    builder.AddMeter(serviceSettings.ServiceName)
          .AddHttpClientInstrumentation()
          .AddAspNetCoreInstrumentation()
          .AddPrometheusExporter();
  });
}
```

9. Run all microservices

10. Perform a few purchases in Frontend portal

11. Verify Trading metrics keep reporting

12. Show the new MassTransit metrics

13. Commit and push changes

In the next lesson you will deploy Prometheus to your AKS cluster.