Hands■On Lab: Serilog + OpenTelemetry → Azure Application Insights (with KQL)

Goal: instrument a .NET Aspire-style service using Serilog and OpenTelemetry, publish telemetry to Azure Application Insights/Log Analytics, and query it with KQL. Optional: run in Azure Container Apps and wire CI with GitHub Actions.

1) Prerequisites

- Azure subscription with permissions to create Resource Group, Application Insights, and Log Analytics Workspace.
- CLI: Azure CLI (az), .NET SDK 8+, Docker (optional for ACA), Git.
- NuGet: Serilog.AspNetCore, Serilog.Sinks.Console, Serilog.Enrichers.Environment, Serilog.Enrichers.Process, Serilog.Enrichers.Thread, Serilog.Sinks.ApplicationInsights (optional), OpenTelemetry.Extensions.Hosting, OpenTelemetry.Exporter.AzureMonitor, OpenTelemetry.Instrumentation.AspNetCore, OpenTelemetry.Instrumentation.Http, OpenTelemetry.Instrumentation.SqlClient.

2) Azure Setup (App Insights + Log Analytics)

```
# variables
rg=rg-telemetry-lab
loc=westeurope
law=law-telemetry-lab
appi=appi-telemetry-lab

# create resource group
az group create -n $rg -l $loc

# create log analytics workspace
az monitor log-analytics workspace create -g $rg -n $law -l $loc

# create application insights (workspace-based)
az monitor app-insights component create -g $rg -a $appi -l $loc --workspace $law

# get connection string (use in app settings or Key Vault)
az monitor app-insights component show -g $rg -a $appi --query connectionString -o tsv
```

3) .NET Web API – add Serilog and OpenTelemetry

Program.cs (minimal example)

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;
using Serilog;

var builder = WebApplication.CreateBuilder(args);

// ---- Serilog (console + enrichers; optional AI sink) ----
Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(builder.Configuration)
    .Enrich.WithEnvironmentName()
    .Enrich.WithProcessId()
```

```
    .Enrich.WithThreadId()
    .WriteTo.Console() // structured JSON if configured
    .CreateLogger();
builder.Host.UseSerilog();

// ---- OpenTelemetry ----
var serviceName = "TelemetryLab.Api";
var serviceVersion = "1.0.0";

builder.Services.AddOpenTelemetry()
    .ConfigureResource(r => r.AddService(serviceName: serviceName, serviceVersion: serviceVersion
    .WithTracing(tracer => tracer
        .AddAspNetCoreInstrumentation()
        .AddHttpClientInstrumentation()
        .AddSqlClientInstrumentation(o => { o.SetDbStatementForText = true; })
        .AddAzureMonitorTrace()) // exports to App Insights
    .WithMetrics(metrics => metrics
        .AddAspNetCoreInstrumentation()
        .AddRuntimeInstrumentation()
        .AddHttpClientInstrumentation()
        .AddAzureMonitorMetric());

builder.Services.AddControllers();

var app = builder.Build();
app.MapGet("/healthz", () => Results.Ok(new { status = "ok" }));
app.MapGet("/demo", () =>
{
    Log.Information("Processing /demo at {UtcNow}", DateTime.UtcNow);
    return Results.Ok(new { message = "Hello, telemetry!" });
});
app.MapControllers();

app.Run();
```

appsettings.json (Serilog + Azure Monitor connection string)

```
{
  "Serilog": {
    "Using": [ "Serilog.Sinks.Console" ],
    "MinimumLevel": "Information",
    "WriteTo": [
      { "Name": "Console", "Args": { "formatter": "Serilog.Formatting.Compact.CompactJsonFormatte
    ],
    "Enrich": [ "FromLogContext", "WithMachineName", "WithThreadId" ],
    "Properties": { "Application": "TelemetryLab.Api" }
  },
  "AzureMonitor": {
    "ConnectionString": "InstrumentationKey=<auto>;IngestionEndpoint=https://westeurope-5.in.appl
  }
}
```

Tip: Prefer storing the Azure Monitor connection string in Key Vault and load via Managed Identity in production.

4) Run & Test Locally

- Run the API: `dotnet run`
- Hit endpoints: `curl https://localhost:5001/demo` and `/healthz`
- Confirm Serilog output in console (JSON events) and verify OTEL exporter sends telemetry (look for 'Exported X spans' logs)

5) Query Telemetry with Kusto (KQL)

```
// Requests (HTTP) with latency > 300ms
requests
| where timestamp > ago(1h)
| where cloud_RoleName == "TelemetryLab.Api"
| where duration > 300ms
| project timestamp, name, resultCode, duration, operation_Id

// Serilog custom events (if routed via AI sink or OTEL logs)
traces
| where timestamp > ago(1h)
| where cloud_RoleName == "TelemetryLab.Api"
| where message has "Processing /demo"
| project timestamp, severityLevel, message, customDimensions

// Dependency calls (SQL, HTTP)
dependencies
| where timestamp > ago(1h)
| where cloud_RoleName == "TelemetryLab.Api"
| project timestamp, name, target, type, duration, success
```

6) Optional: Containerize & Deploy to Azure Container Apps

```
# Dockerfile
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 8080

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY . .
RUN dotnet publish -c Release -o /out

FROM base AS final
WORKDIR /app
COPY --from=build /out .
ENV ASPNETCORE_URLS=http://+:8080
ENTRYPOINT ["dotnet", "TelemetryLab.Api.dll"]
```

```
# Create ACA environment & app (simplified)
acae=acae-telemetry-lab
ca=ca-telemetry-lab
acr=acrtelemetrylab123

# create ACR + push image
az acr create -g $rg -n $acr --sku Basic
az acr login -n $acr
docker build -t $acr.azurecr.io/telemetrylab:latest .
docker push $acr.azurecr.io/telemetrylab:latest

# ACA env and app (assumes MI permissions to App Insights if needed)
az containerapp env create -g $rg -n $acae -l $loc
az containerapp create -g $rg -n $ca --environment $acae \
  --image $acr.azurecr.io/telemetrylab:latest \
  --target-port 8080 --ingress external \
  --env-vars "AzureMonitor__ConnectionString=$(az monitor app-insights component show -g $rg -a $
```

7) Optional: GitHub Actions – Build, Push, Deploy

```
name: deploy-telemetry-lab
on:
```

```
  workflow_dispatch:
permissions:
  id-token: write
  contents: read
jobs:
  build-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-dotnet@v4
        with: { dotnet-version: '8.0.x' }
      - name: Build
        run: dotnet build -c Release
      - name: Azure login (OIDC)
        uses: azure/login@v2
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
      - name: ACR build & push
        run: |
          az acr login -n ${{ vars.ACR_NAME }}
          docker build -t ${{ vars.ACR_NAME }}.azurecr.io/telemetrylab:${{ github.sha }} .
          docker push ${{ vars.ACR_NAME }}.azurecr.io/telemetrylab:${{ github.sha }}
      - name: Deploy to ACA
        run: |
          az containerapp update -g ${{ vars.RESOURCE_GROUP }} -n ${{ vars.CA_NAME }} \
            --image ${{ vars.ACR_NAME }}.azurecr.io/telemetrylab:${{ github.sha }}
```

8) Troubleshooting

- Nothing in App Insights? Verify the Azure Monitor connection string and outbound internet from container.
- Traces visible but logs missing? Ensure you’re exporting OTEL traces and (optionally) OTEL logs or using Serilog AI sink.
- High cardinality customDimensions can raise costs; normalize where possible.
- Sampling: Azure Monitor enables sampling; adjust if you’re missing events during load tests.
- In ACA, confirm revisions are healthy and probes pass (/healthz).