Adding Logging

(Demo prep)

- Reset all federated credentials in AAD to use dotnetmicroservices organization
- Delete SEQ from AKS cluster, plus all mappings
- Point the NuGet source to dotnetmicroservices organization

Start

In ASP.NET Core you can make use of the ILogger interface to quickly add logs to your microservice application code. Let's see how we can update our Trading microservice to emit logs for any new purchase operation.

In Trading repo

1. Add logging to PurchaseController:

```
public class PurchaseController : ControllerBase
{
    …
    private readonly IRequestClient<GetPurchaseState> purchaseClient;
    private readonly ILogger<PurchaseController> logger;

    public PurchaseController(
        IPublishEndpoint publishEndpoint,
        IRequestClient<GetPurchaseState> purchaseClient,
        ILogger<PurchaseController> logger)
    {
        …
        this.purchaseClient = purchaseClient;
        this.logger = logger;
    }

    …

    [HttpPost]
    public async Task<IActionResult> PostAsync(SubmitPurchaseDto purchase)
    {
        var userId = User.FindFirstValue("sub");

        logger.LogInformation("Received purchase request of item " + purchase.ItemId + " from user " + userId);

        …
    }
}
```

2. Change default logging level to Information in appsettings.Development.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      …
    }
  },
  …
}
```

3. Run all microservices and try a purchase

4. Explain the logs and the log category

5. Enable Json logging in Program.cs:

```
using Microsoft.Extensions.Logging;

public class Program
{
    …

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            …
            .ConfigureWebHostDefaults(webBuilder =>
            {
                …
            })
            .ConfigureLogging(builder => builder.AddJsonConsole(options =>
            {
                options.JsonWriterOptions = new JsonWriterOptions
                {
                    Indented = true
                };
            }));
}
```

6. Run all microservices and try a purchase again

7. Explain structured logging

8. Use structured logging in PurchaseController:

```
public class PurchaseController : ControllerBase
{
   …

   [HttpPost]
   public async Task<IActionResult> PostAsync(SubmitPurchaseDto purchase)
   {
      var userId = User.FindFirstValue("sub");

      logger.LogInformation(
         "Received purchase request of {Quantity} of item {ItemId} from user {UserId} with CorrelationId {CorrelationId}...",
         purchase.Quantity,
         purchase.ItemId,
         userId,
         purchase.IdempotencyId);
      …
   }
}
```

9. Run all microservices and try a purchase again

10. Show structured logs

11. Add logging to PurchaseStateMachine:

```
public class PurchaseStateMachine : MassTransitStateMachine<PurchaseState>
{
   private readonly MessageHub hub;
   private readonly ILogger<PurchaseStateMachine> logger;

  …

   public PurchaseStateMachine(MessageHub hub, ILogger<PurchaseStateMachine> logger)
   {
      …
      this.hub = hub;
      this.logger = logger;
   }
  …
   private void ConfigureInitialState()
   {
      Initially(
         When(PurchaseRequested)
            .Then(context =>
            {
               …
               context.Instance.LastUpdated = context.Instance.Received;
               logger.LogInformation(
                  "Calculating total price for purchase with CorrelationId {CorrelationId}...",
                  context.Instance.CorrelationId);
```

```
        })
        …
        .Catch<Exception>(ex => ex.
          Then(context =>
          {
            …
            context.Instance.LastUpdated = DateTimeOffset.UtcNow;
            logger.LogError(
              context.Exception,
              "Could not calculate the total price of purchase with CorrelationId {CorrelationId}. Error: {ErrorMessage}",
              context.Instance.CorrelationId,
              context.Instance.ErrorMessage);
          })
          …
    );
}

private void ConfigureAccepted()
{
    During(Accepted,
      Ignore(PurchaseRequested),
      When(InventoryItemsGranted)
        .Then(context =>
        {
          context.Instance.LastUpdated = DateTimeOffset.UtcNow;
          logger.LogInformation(
            "Items of purchase with CorrelationId {CorrelationId} have been granted to user {UserId}. ",
            context.Instance.CorrelationId,
            context.Instance.UserId);
        })
        …
        .Then(context =>
        {
          …
          context.Instance.LastUpdated = DateTimeOffset.UtcNow;
          logger.LogError(
            "Could not grant items for purchase with CorrelationId {CorrelationId}. Error: {ErrorMessage}",
            context.Instance.CorrelationId,
            context.Instance.ErrorMessage);
        })
        …
      );
}

private void ConfigureItemsGranted()
{
    During(ItemsGranted,
      Ignore(PurchaseRequested),
      Ignore(InventoryItemsGranted),
      When(GilDebited)
        .Then(context =>
        {
          context.Instance.LastUpdated = DateTimeOffset.UtcNow;
          logger.LogInformation(
```

```
                "The total price of purchase with CorrelationId {CorrelationId} has been debited from user {UserId}. Purchase complete.",
                context.Instance.CorrelationId,
                context.Instance.UserId);
            })
            …
        When(DebitGilFaulted)
            …
            .Then(context =>
            {
                …
                context.Instance.LastUpdated = DateTimeOffset.UtcNow;
                logger.LogInformation(
                "Could not debit the total price of purchase with CorrelationId {CorrelationId} from user {UserId}. Error: {ErrorMessage}.",
                context.Instance.CorrelationId,
                context.Instance.UserId,
                context.Instance.ErrorMessage);
            })
            …
    );
  }
  …
}
```

12. Run all microservices and try a purchase

13. Notice the Trading logs

14. Undo the changes to Program.cs

15. Commit and push

In the next lesson you will learn how to stand up a Seq server in your box via the Seq docker image.