Handling partial failure

(Demo prep)

Implementing exponential backoff

1. In the last lesson we introduced a 1 second timeout policy to our inventory microservice. Let's now use the retries with exponential backoff technique to let Inventory retry the calls to Catalog a few times before giving up.

2. Update Startup.ConfigureServices:

```
services.AddHttpClient<CatalogClient>(client =>
{
   client.BaseAddress = new Uri("https://localhost:5001");
})
.AddTransientHttpErrorPolicy(builder => builder.Or<TimeoutRejectedException>().WaitAndRetryAsync(
   5,
   retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)),
   onRetry: (outcome, timespan, retryAttempt) =>
   {
      var serviceProvider = services.BuildServiceProvider();
      serviceProvider.GetService<ILogger<CatalogClient>>()?
         .LogWarning($"Delaying for {timespan.TotalSeconds} seconds, then making retry {retryAttempt}");
   }
))
.AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
```

3. Restart both services

4. Try the query again

5. Add the jitterer:

```
Random jitterer = new Random();

services.AddHttpClient<CatalogClient>(client =>
{
   client.BaseAddress = new Uri("https://localhost:5001");
})
.AddTransientHttpErrorPolicy(builder => builder.Or<TimeoutRejectedException>().WaitAndRetryAsync(
   5,
   retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt))
            + TimeSpan.FromMilliseconds(jitterer.Next(0, 1000)),
   onRetry: (outcome, timespan, retryAttempt) =>
```

```
    {
        var serviceProvider = services.BuildServiceProvider();
        serviceProvider.GetService<ILogger<CatalogClient>>()?
            .LogWarning("Delaying for {delay} seconds, then making retry {retry}", timespan.TotalSeconds,
retryAttempt);
    }
))
.AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
```

6. Restart both services

7. Try the queries again

This is much better and would certainly help us handle temporal glitches when reaching out to Catalog service. However, image that a widespread network outage is preventing us from reaching Catalog, or perhaps is preventing Catalog from reaching its dependencies. Such outage will likely last at least half an hour and if we have all our Inventory service threads retrying that much time we may end up exhausting its resources.

In the next lesson we will learn more about resource exhaustion and how to deal with it via the circuit breaker technique.