# EmpowerID - Lab Exercise Technical Decisions, Trade-offs, and Challenges

1. **SQL Server and database design**

I created an Azure SQL server database resource for this lab. It was secured enough because of the IP firewall rule where I only put my IP address to have an access, as well as Azure resources. We can still further limit the access of the server by adding private endpoint. I created two databases, one serves as primary for the online transaction database, while the others serve as Change data capture database.

The database schema was just simple, I implemented primary keys, foreign keys, and dates data constraints to the table designs. I added quantity and amount in the orders tables which I think is better because we will be needing them later when we do our enhanced search and calculations using GPT custom model. This is an OLTP database, so I only used the needed index, the primary key will be enough since it will automatically create an index. This can be verified when running the query execution plan and reviewing extended events.

Enabling CDC in the primary database and target tables was fine. We can immediately see the CDC tables populated with the changes. However, enabling default CDC means CDC tables will reside in similar database. There’s a caveat on it, additional database performance overheads and worse if there is database failure where both primary and CDC data might lose.

The requirement on this lab is to create a CDC on separate database. There are great 3rd party applications out there for this job however due to pricing and some issues I did not resort to them. Fortunately, Azure Data Factory just launched a preview version of Change data capture. I created a CDC pipeline to automatically monitor the changes in the primary database and put the changes on a separate CDC database.

For the test data, I used Redgate SQL Data Generator to generate data. It automatically detects the SQL server schema and apply the needed data. We can also easily customize it; we can define the minimum and maximum value on a specific column. It can also set the distribution with either random or incremental. I was able to generate my test data in just few seconds, after which a report is also generated to verify that the test was successful written to the target database.

1. **Data Factory**

A great tool for ELT and scaling out serverless data integration and transformation. I have been using this tool for multiple projects and I am amazed for its features and usage. We can use integration agent to securely connect to any data sources (clouds or on-prem) with different data type connection and file support.

I created linked services to seamlessly connect to MSSQL server and blob storage (where I put sample database data). I also created a key vault to store database credential. These connections were used by the pipelines I created for the CSV database sources targeting products, categories, and orders tables. These pipelines are available in C# web page thru REST Api connections. They can be triggered anytime from the C# web page to start the ETL process. Logs can also be viewed in the web page for monitoring purposes.

Change Data Capture, capturing separate database is done through Data factory. Although this is still in preview mode and with known limitations. I still believed this is an alternative way of implementing CDC outside of the 3rd party applications. You can set the CDC run to real-time to capture the changes immediately or every schedule defined in the CDC. Since this is still in preview mode, there were some cases where I expected to capture the changes however CDC run logs did not capture any.

1. **C# Code**

On top of the requirements, here’s a brief definition of the C# lab project.

Language /Platform: C# .Net Core

Architecture pattern: Model-View-Controller (MVC)

.Net Framework: .Net 6.0

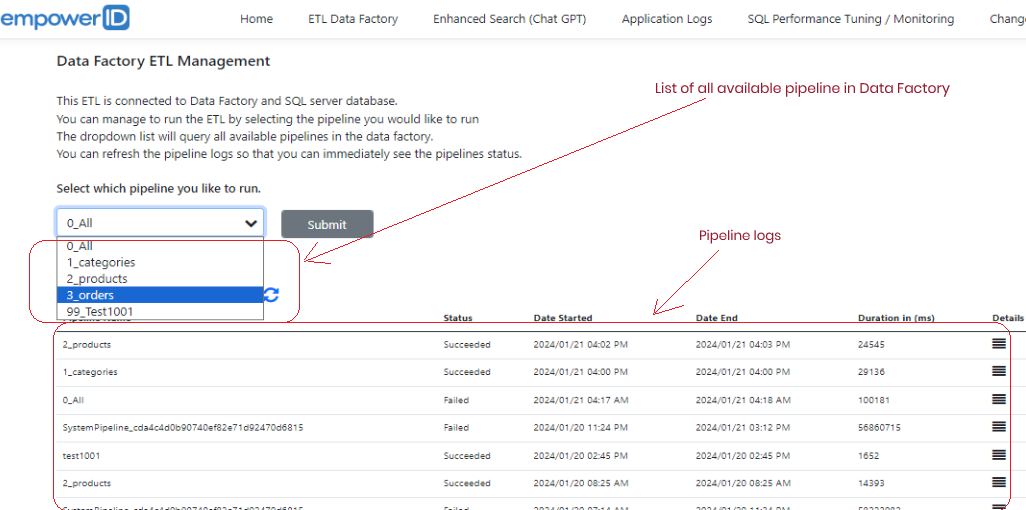
Logging: Default .net core logging, with file system, blob, and insights logging enabled.

Configuration: Environment variables

API Usage: Azure management Rest API and Chat GPT API

**Requirement**: Implementing CLI that allows the user to initiate and monitor ETL pipelines.

Exposing CLI to user might have caveat especially when user is not proficient enough to CLI commands. Hence, I decided not to expose CLI command and had tried different approach. Azure has management REST API for data factory. These APIs can query the data factory pipelines and can manage to execute them. With these available features, I just exposed a parameterized dropdown list containing all pipelines where a user can easily choose and submit and run the ETL process. I added a feature where we can see the latest logs (3 days logs). Clicking refresh icon is also available so that we can immediately check the status of the pipeline we run.



**Requirement**: Search the data using Azure Cognitive search and search the product with all available parameters.

Azure cognitive provides secure information retrieval at scale over user-owned content in traditional and conversational search applications. However, I still believe that a combined Cognitive and OpenAI feature will be superior. Unfortunately, Azure did not make OpenAI available yet on its resources. Although it can be available by request.

A screenshot of a computer

Description automatically generated

With that, I opt to used OpenAI main portal (not Azure OpenAI). Chat GPT has new available feature called My GPTs or Assistant in developer portal. It is where we can create our own custom GPT / GPT assistant on top of the GPT based model. I created an instruction for the custom GPT and then uploaded the data / CSV files for ingestion. Using the REST API, GPT key and the assistant key, I was able to connect to the custom GPT and make a conversation using the C# code in the web page. I manually trained the model and might need more time for learning.

Enhanced Search - Chat GPT Custom Model

A screenshot of a computer

Description automatically generated

**Requirement**: Code to handle errors and exceptions and log to application events.

My C# web project is capable of handling errors and exceptions and will be stored in file system, blob storage and application insight. For demo purposes, all these logging are enabled in the project.

File system logging – aside from enabling file system logging in the code, Azure web app file system logging must also be enabled, to capture and store the logs on its internal system.

Blob logging – my C# is also blob enabled. Like file system logging, Azure blob logging must also be enabled in the Azure web app, set the blob container, retention, and file size.

App Insight – this logging is also enabled on my C# code. App Insight must also be turned on in the Web app so that logs will be directed to App Insight.

Azure web app also provides stream logging, this is great specially when we want a real-time check on the logs.

**C# code logging enabled for file system, blob storage, and app insight.**

A screenshot of a computer program

Description automatically generated

**Azure Web App logging enabled / turned-on.**

A screenshot of a computer

Description automatically generated

**Requirement**: Code to optimize SQL queries and database design for performance and scalability.

My C# code does not query the database hence the query optimization may not apply. Although I have evaluated some queries vs my database design. Please refer to #1 Database and database design for additional information.

Below are just some the tools I use to optimized and monitor the SQL server database.

* SQL Server Profiler for On-Prem database.

A screenshot of a computer

Description automatically generated

* Execution Plan and live query statistics

A screenshot of a computer

Description automatically generated

* Extended Events for Azure database and On-prem database

A screenshot of a computer

Description automatically generated

* Azure SQL Database Monitoring.

A screenshot of a computer

Description automatically generated

**Requirement**: Code to implement CDC to capture specific changes to the data in the database and write them out to another database.

CDC implementation at application might not be feasible. Because data manipulation does not only happen on this application. Data changes can be from SSMS, terminals, external applications among others. I enabled native CDC on the primary database and enabled CDC for the tables. However, the requirement is to apply CDC at separate database. I unable to use 3rd party software to implement CDC for pricing / usage issue. Fortunately, Data Factory has Change data capture feature. This has been implemented in data factory, please read #2 Data Factory.

* Data Factory Change data capture

A screenshot of a computer

Description automatically generated

1. **Key vault**

Used for storing passwords / credentials / certificates.

Author: Erwin Caras

Purpose: This document is intended for EmpowerID lab exercise technical decisions and trade-offs.

Classification: Internal-Only

Date created: January 16, 2024

Date revision: January 22, 2024

Doc Version: v1.0.0.3