**The Problem**

Programmatic support in Azure Data Factory:

IMA needs a place to run code!

Ideas:

* Azure Batch
* Azure Container Instances
* Azure Functions
* Virtual Machine

In all of these cases, we want to enable metadata driven executions of ETL/Programmatic processes. These main use cases would be:

1. Extractions from API’s where ADF Copy activities cannot be used to effectively extract the entire dataset with as little effort as possible.
2. Transformations that can be performed on a single node, to serve as a replacement for distributed compute on ADF spark clusters. Would serve as an alternative and cost effective ETL pattern.
3. Any other operation that might benefit from being configured through a code-based environment.

As part of this development, we will want to be paying attention to a couple of things.

* Complexity: The solution should be easy to understand
* Local Development possible: It should be possible to easily develop solutions for the new platform in a local environment where local compute can be used to accelerate development velocity.
* DevOps considerations: Any code that we develop to run on this solution should be easy to build and deploy in a repeatable manner. It should enable unit testing and provide some interlock with our Azure Dev Ops environment.
* Security: Such a solution should be feasible in our current Azure ecosystem, an environment where the expectation is that the service will be network isolated. Ie the solution should live within an Azure Vnet that cannot be reached from public networks.
* Interoperability with Azure Data Factory: The expectation is that the solution will receive requests from Azure Data Factory using metadata driven requests. It should be able to provide a synchronized response to the service and share key metadata facts including error messages, row counts etc.
* Clear insight into cost of use.
* Scalability: Such a solution should be capable of being extended to run any process that a single node machine can. Parallelization should be front of mind. It should be straightforward to set up multiple nodes, while aligning with the level of complexity our team is comfortable with.

**A Proof of Concept**

For all these compute platforms I will demonstrate how the technology maps to our use case. This can be done through a set of requirements that closely matches our use case:

1. Extract from an API and write to blob storage in a time efficient manner relative to what ADF can accomplish.
2. Perform an end-to-end Transformation and Loading of a dataset to a database. Datasets should be extracted as parquet from ADLS, then written to a relational database. Suppose this process supports a table with about 1 million rows (match to a larger but not largest EDA table). Test for full and incremental pattern.
3. Require the solution to be capable of running many operations in parallel (let’s say 100 concurrent API reads and 15 concurrent tables). Demonstrate scalability and flexibility.

All of these operations will be developed in Python. I will use standard API libraries for the test extraction on a publicly accessible API testing endpoint. I will use best in class in-memory data processing libraries for ETL processing (Polars, Pandas, DuckDb). Ideally I will demonstrate that ETL transformations can be written as SQL for easy lift and shift at some future point into an ELT tool (Snowflake/BigQuery/Databricks). I will use standard azure libraries to interface with azure services. For a datastore I will most likely spin up a small Azure SQL or Postgre box. I will look for test datasets already available as parquet, and if not, I will create my own dummy datasets.

**The Concepts locally:**

I will start this off by getting the tests all set up and running locally on my personal machine. I will create a git repository specific to this effort to iterate on my solution. This is in a repo called ComputePOC. I will publish to githhub if need for any of the compute deployment operations. There will be no sensitive information stored in the repo.

API Test Extraction:

* I figured I could extract time series data from some FRED datasets, as their API is publicly accessible. My api Key is 0c34b5e00a2480151931f7c6fcc6fe5c.
* Performance/concurrency is hardly a consideration, the API has a limit of 120 requests per minute
* I will sample the full dataset. I will pull 50 releases, and 50 series for each release. I will pull the full set of data for each series I extract
* I will write the releases/series/observations to ADLS blob storage

ETL Test

* I will combine the full set of extracted FRED API data into a single big table
* As there are many source files, I will need a performant way to read the files from ADLS
* I will write the transformation as SQL in duck DB.
* I will output the file back to ADLS as a single parquet file.

T