Building a budgeting/ forecasting system in Microsoft business apps

MICHAEL HAUPTMAN

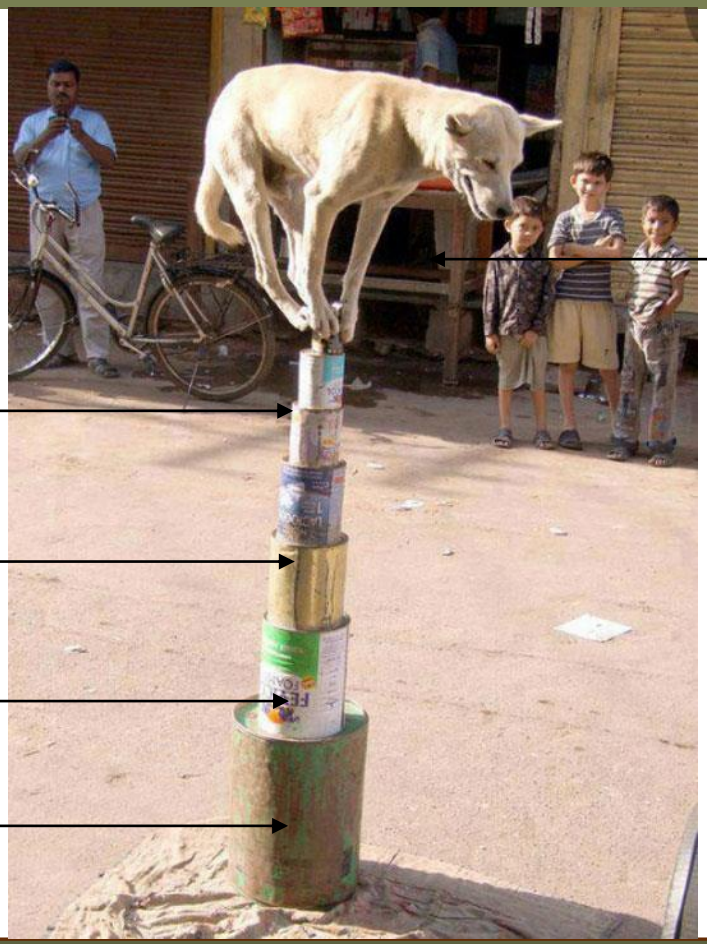
SYDNEY MICROSOFT BUSINESS APPS COMMUNITY
4TH JUNE 2024

Aim: share some powerful but less well documented features of the Microsoft stack

Agenda

- Overview of the budgeting/forecasting project
  - Motivation and requirements
  - Solution and architecture
- Demonstration of the system
- Discussion of useful features
  - Dataverse APIs and interfaces
  - Fabric Lakehouse and delta tables
  - Miscellaneous – editable tables, TDS endpoint, Dataverse/Fabric integration

Some disclaimers- I am a self-taught developer



My code

Chat-GPT

Stack Overflow

Microsoft Learn

Udemy

This has been a personal project. I am a team of one

Budgeting/forecasting tool appears to be well suited to the Microsoft stack …

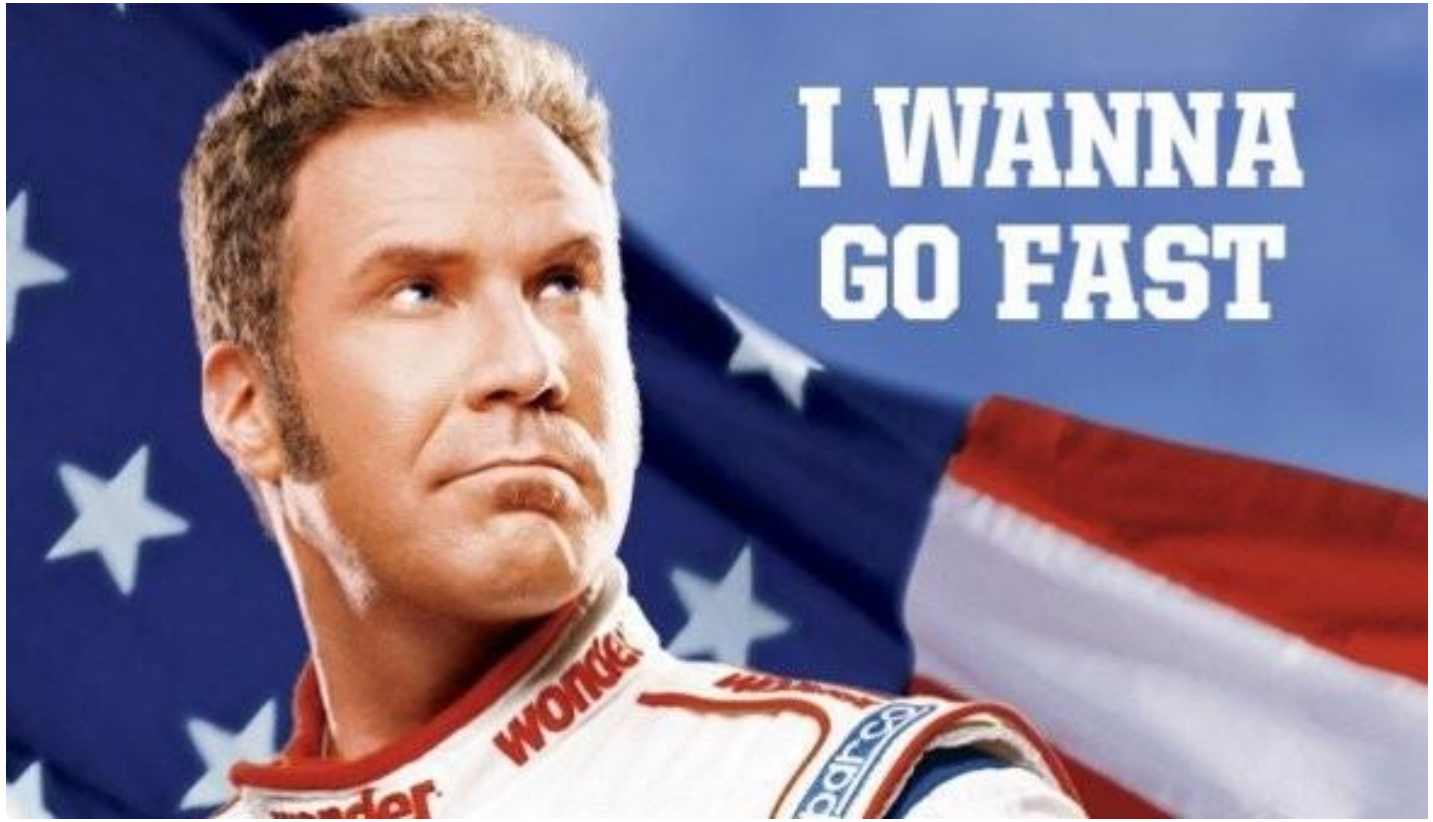
Current budgeting/planning systems:

- A constellation of Excel spreadsheets (shudder!)
- Purpose-built tools: Anaplan, Jedox, Workday Adaptive Planning
- Sometimes use in-memory processing, which can require work-arounds with large models

Relevant features of the Microsoft stack:

- ✓ User access control, role-defined access, row level security, audit of data changes
- ✓ Approval process
- ✓ Incorporate data from many different sources
- ✓ Application Lifecycle Management
- ✓ Ability to incorporate AI
- ✓ Ability to rapidly process large data sets

Ability to rapidly process large data sets

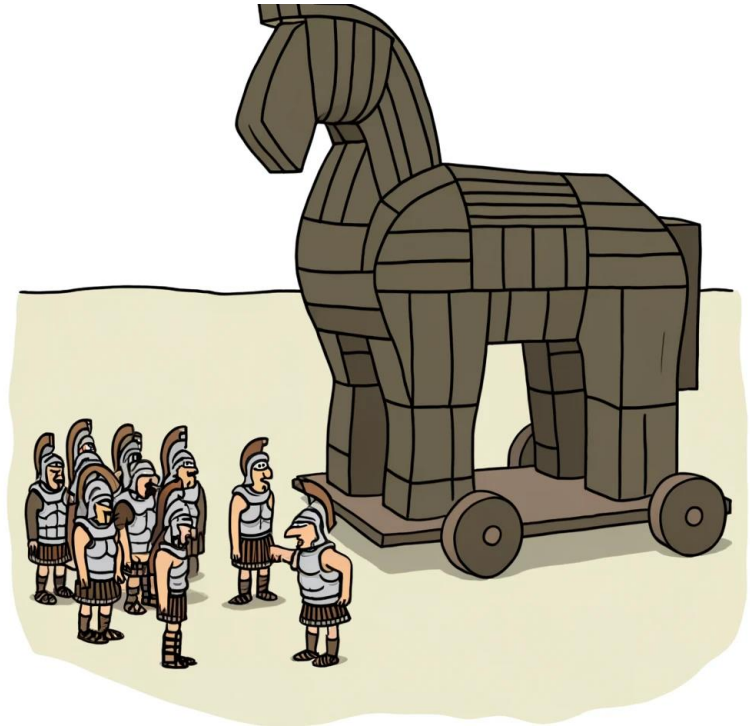


Handle models with thousands of rows of input, with forecast periods of 20 years+

- Seek fast calculation solutions
- Seek fast read/write solutions

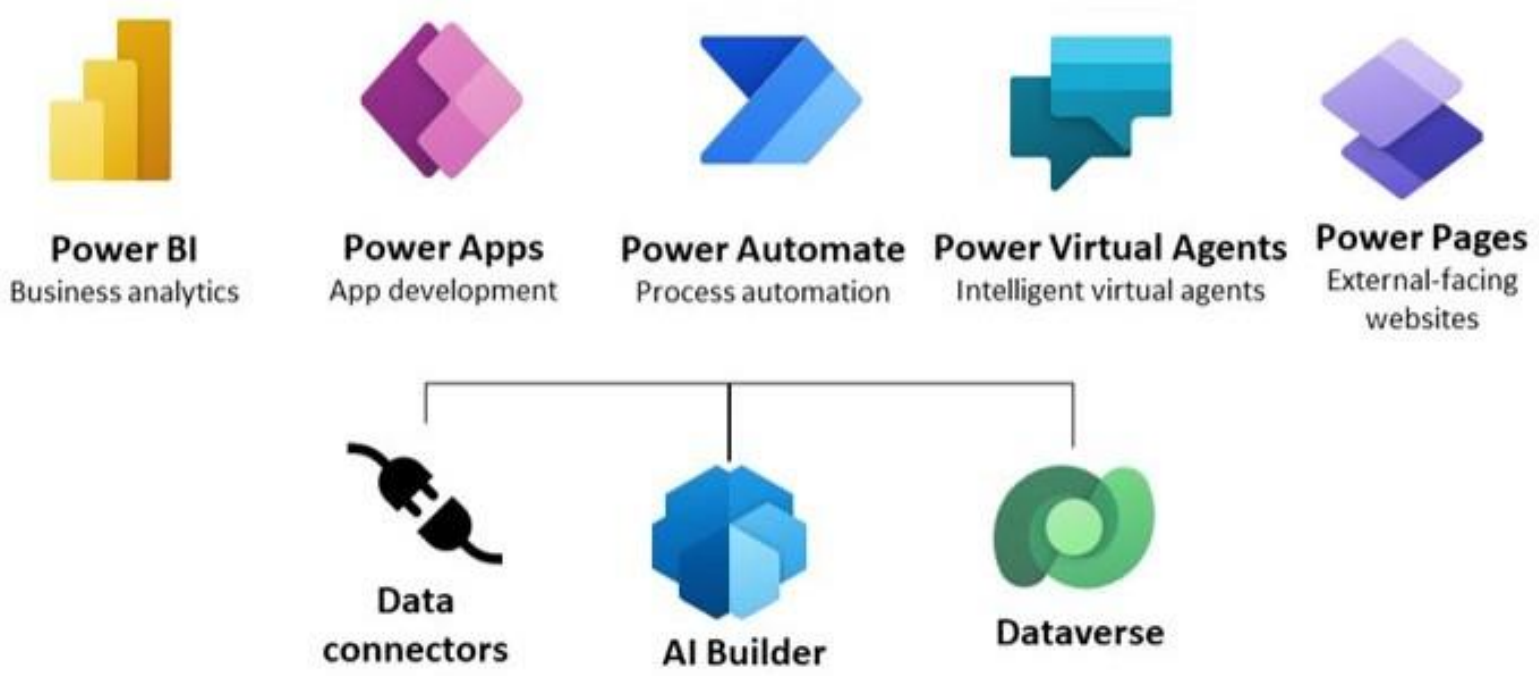
Goal: *< 10 mins from input to forecast*

Many of you will be familiar with Power Platform, Azure and Fabric

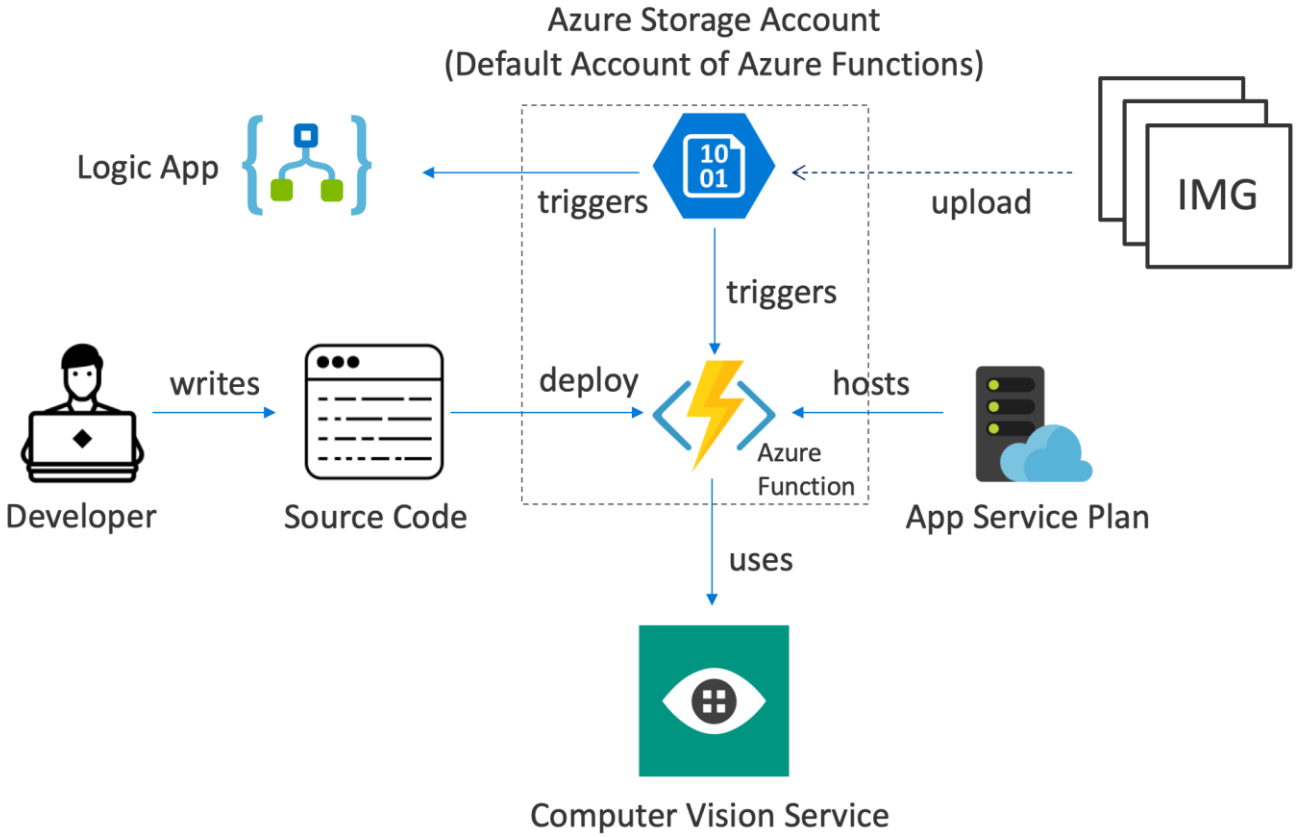


"For some of you men, this will be your first time in a horse"

Overview of Power Platform



Overview of Azure Functions (serverless processing)



Overview of Fabric

Design choices for my budget/forecast tool

Area	Requirements	Design choice	Rationale
"Front end" data entry	• User access control, role-based access, audit etc • Spreadsheet-like data entry	**Power App Model Driven App** (not Canvas App)	• Optimised for integration with Dataverse tables • Editable table
Calculations and data manipulation	• Fast response times • Multiple concurrent users	**Python program in Azure Function** (not Spark Notebook in Azure)	• Avoid overhead to "spin up" a Spark job • Configure as "microservices" rather than monolithic program • Cost effective – only really charged when run
Storage of calculations and final reporting	• Fast data loading and saving • Attractive and fully featured reporting system ,with drill-downs etc	**Fabric Lakehouse/ Delta Table for storage** (not Dataverse) **and PowerBI for reporting** (not Power Apps)	• Delta tables enables faster data saving than Dataverse • Integrated with PowerBI • Power Apps has only rudimentary reporting

Architecture

Enter data

PowerApps

Dataverse

A. Configure PowerApps/ Dataverse for data entry

Power Automate

Azure Functions

Power BI

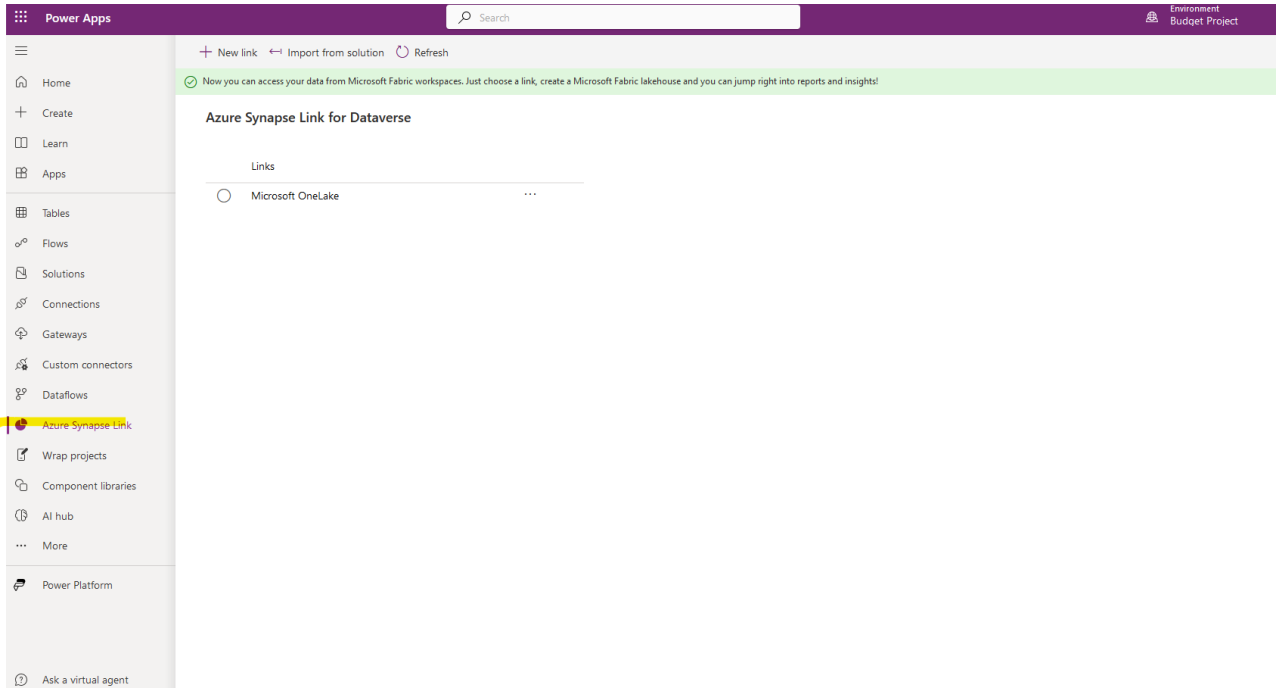
C. Calculate forecast

Fabric Lakehouse

Receive forecast

B. Move data from Dataverse to Fabric Lakehouse (ETL)

A digression – Connecting Dataverse to Fabric Lakehouse



- No extra charge for linking, but may incur extra storage
- 2 different approaches
  1. Link to Fabric
  2. Azure Synapse Link
- Link to Fabric not real time
- Azure Synapse Link can be slow to refresh if too many tables included

https://learn.microsoft.com/en-us/power-apps/maker/data-platform/azure-synapse-link-synapse
https://learn.microsoft.com/en-us/power-apps/maker/data-platform/azure-synapse-link-view-in-fabric
https://www.serverlesssql.com/fabric-link-for-dataverse-whats-in-the-box/

Python choices

Module	Purpose	Description
Polars	• Get data from Dataverse/Lakehouse • Filter and sort it as needed • Load data back to Dataverse/Lakehouse	• “Polars is a blazingly fast DataFrame library for manipulating structured data. The core is written in Rust,” • Multiple core, parallel – but single machine • Handles datasets much larger than available RAM. • Compared to pandas, it can achieve more than 30x performance gains • Supports delta table format
Numpy	• All numerical calculations required for forecast	• “NumPy is the fundamental package for scientific computing with Python” • Numpy core written in C++

Demo and speed test

Headcount and staff expenses

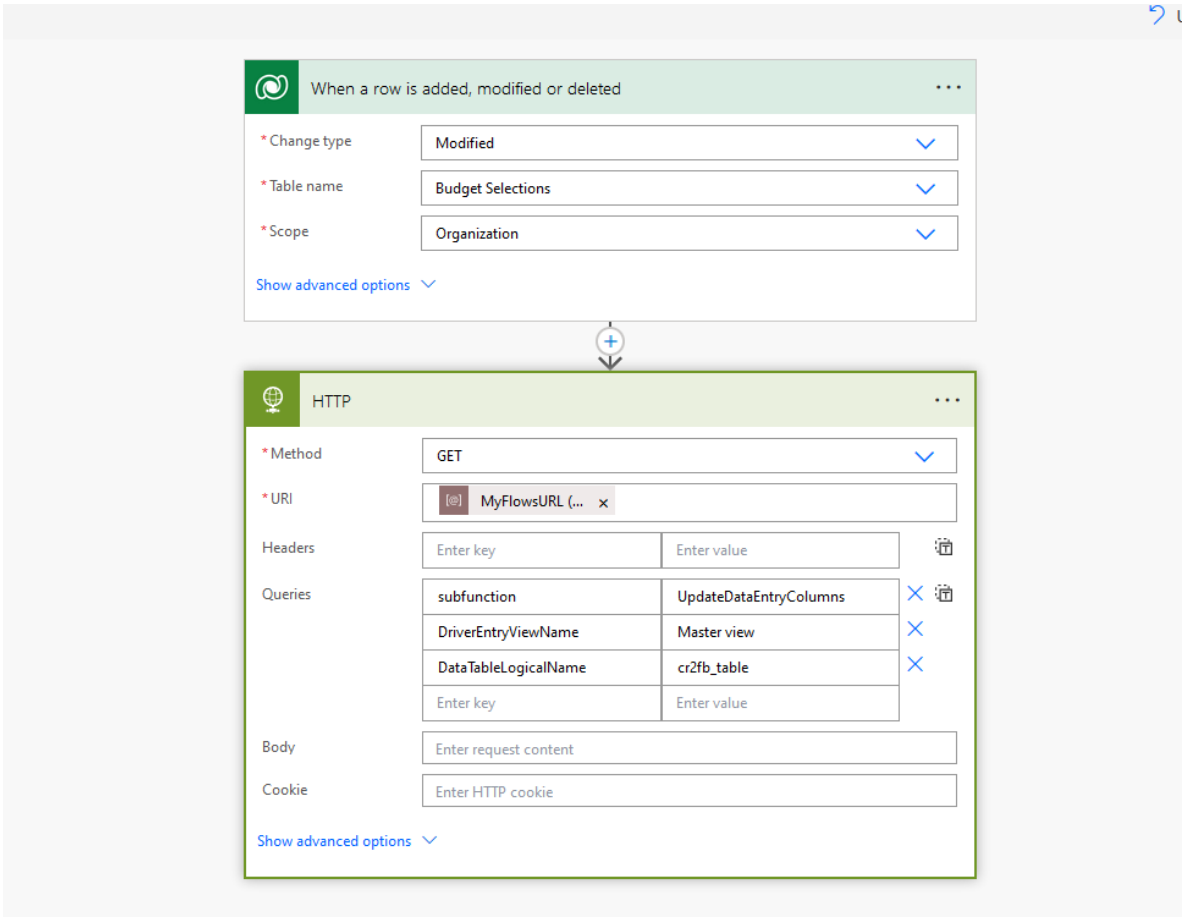
- Based on target headcount and attrition rate, forecasts actual headcount, and salaries, annual leave, superannuation and payroll tax
- Calculates both cash expense and provisions, especially for annual leave
- 1,000 cost centres x 2 scenarios = 2,000 inputs x 240 months

Broker headcount, mortgage sales and portfolio

- Based on current broker headcount, target sales and submission/broker ratio, calculates the number of brokers required
- Cascades the total planned leads to subchannel and lender level, then converts them to submissions and settlements, using conversion rates and delay curves at each stage
- Calculates upfront and trail commission, both cash and accrual
- Forecasts the mortgage portfolio using a retention curve
- 1,000 subchannels x 2 scenarios =2,000 inputs x 240 months (x 5 lenders)


PLEASE STAND BY

Triggering an Azure Function using Power Automate



Dataverse REST API (1): Simple read of Dataverse table data without aggregation

GET
https://org3a8b3a8e.crm.dynamics.com/api/data/v9.2
/mh_costcentres
?$select=mh_costcentreid,mh_cccode,mh_name,mh_cclevel,mh_parencclevel,_mh_parentcc_value,mh_children,_mh_businessunit_value,mh_shortcode

- Can also specify filters, fields from linked tables
- Limit to 5,000 records per request. Note a record is one row of a Dataverse table, so in my budget/forecast model a single record can include 240 months

References:
https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/perform-operations-web-api?source=recommendations

Dataverse REST API (2): Update lots of rows using UpdateMultiple

POST
https://org3a8b3a8e.crm.dynamics.com/api/data/v9.2
/mh_costcentres
/Microsoft.Dynamics.CRM.UpdateMultiple

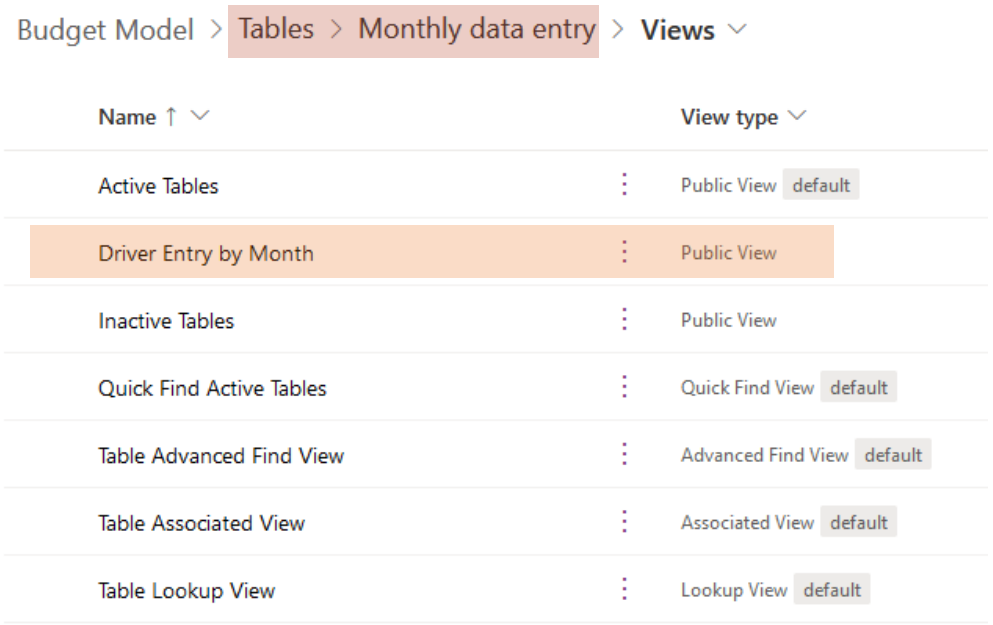
JSON
{"Targets": [
{"mh_costcentreid": “xxx",
"mh_name": "Australian staff -OpCo“
, "mh_children": "true",
"@odata.type": "Microsoft.Dynamics.CRM.mh_costcentre"},
{: ..."}

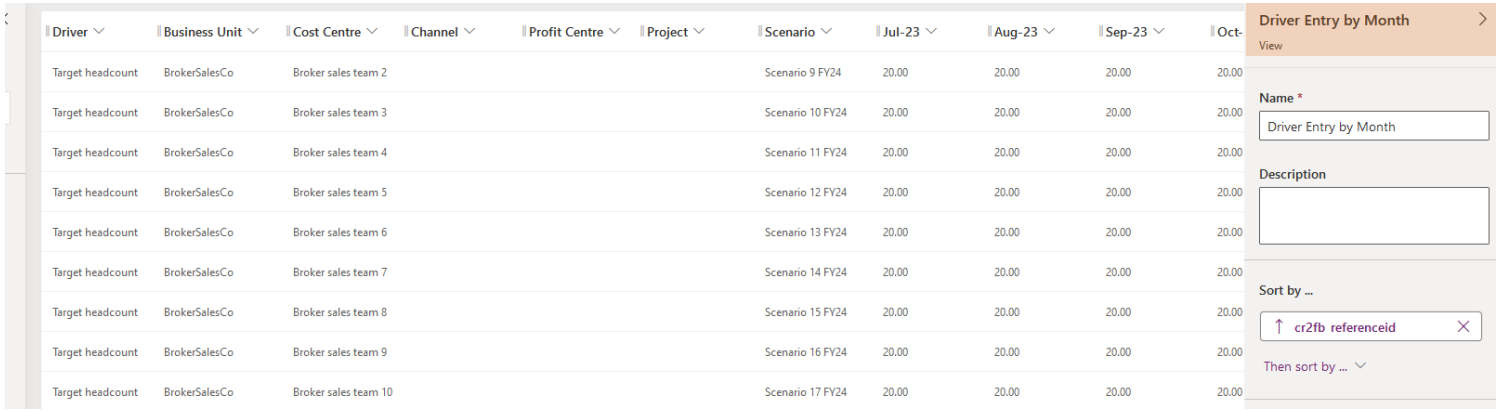
Use *singular* table name

- Can generally use only on custom tables, not system tables
- Limit 500 records per request
- Slow. Loading 50,000 records takes ~ 10-15 mins

References:
https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/reference/updatemultiple?view=dataverse-latest
https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/update-delete-entities-using-web-api

Dataverse REST API (3): Use FetchXML to modify views





https://powerusers.microsoft.com/t5/Microsoft-Dataverse/Creating-a-new-view-using-REST-API/td-p/2630253

Dataverse REST API (3): Use FetchXML to modify views

POST
https://org3a8b3a8e.crm.dynamics.com/api/data/v9.2/savedqueries(e02axxxx4)

Use GUID for your savedqueries view

- Not very well documented
- See Dataverse Rest Builder plugin of XrmToolBox

JSON

```
{'layoutxml':"<grid name=\"resultset\" object=\"10337\"
jump=\"mh_referenceid\" select=\"1\" icon=\"1\"
preview=\"1\"><row name=\"result\"
id=\"mh_globalassumptionsactualid\"><cell
name=\"mh_globalassumption\" width=\"200\"/><cell
name=\"mh_jul23\" width=\"100\"/><cell ...
```

Find reference for your view

https://powerusers.microsoft.com/t5/Microsoft-Dataverse/Creating-a-new-view-using-REST-API/td-p/2630253

Dataverse REST API (3): Use FetchXML to modify views

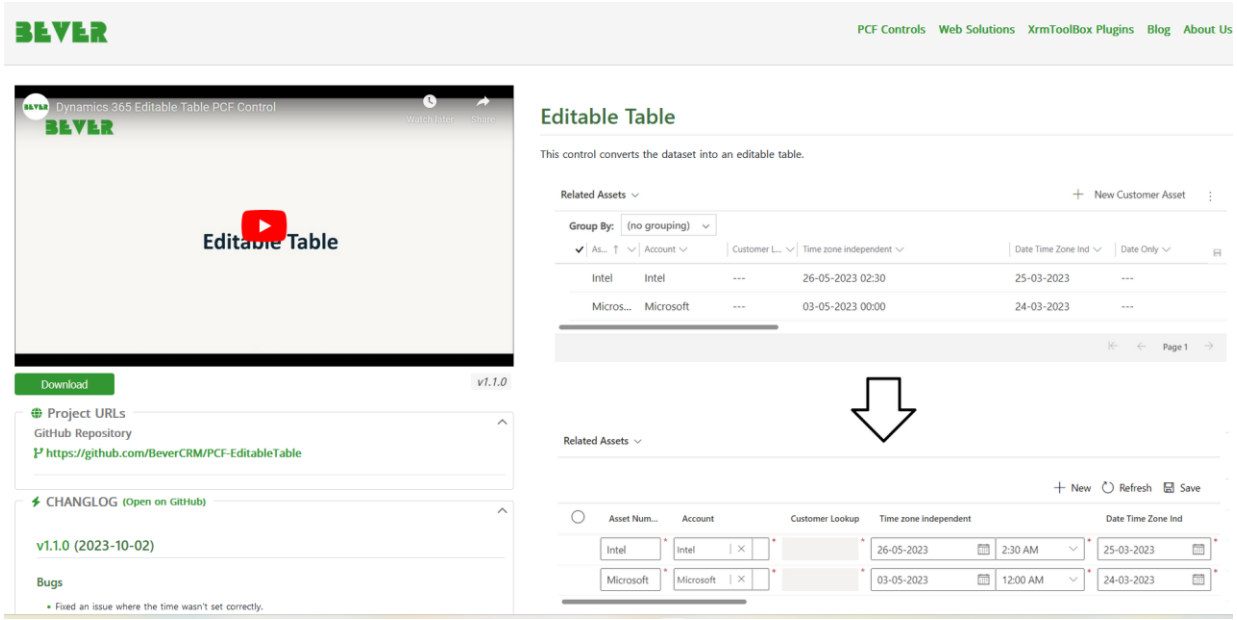
After updating the view table, need to publish to customisation

```
#Post the customisation, i.e. the change to the view
UpdateRequest= DverseUtils.MakeDverseAPIPatchCall(DverseSession=self.DverseSession,request_uri=request_uri,JsonData=JsonData)
#Publish the customation of the LogicalDataTable table
request_uri=f'{self.config["resource"]}/api/data/v9.2//PublishXml'
XMLParameter='{"ParameterXml":"<importexportxml> <entities> <entity>'+self.DataTableLogicalName+'</entity> </entities> </importexportxml>"}'
DverseUtils.MakeDverseAPIPostCall(DverseSession=self.DverseSession,request_uri=request_uri,JsonData=XMLParameter)
```

https://powerusers.microsoft.com/t5/Microsoft-Dataverse/Creating-a-new-view-using-REST-API/td-p/2630253

Digression – Bever Controls Editable Table

The standard Dataverse editable tables control doesn't support new line entry



https://marketplace.bevercrm.com/pcf-controls/7

Dataverse REST API (4): Read of Dataverse table data with aggregation

POST
https://org3a8b3a8e.crm.dynamics.com/api/data/v9.2
/$batch

```
GET /api/data/v9.2/cr2fb_tables?fetchXml=<fetch aggregate="true">
<entity name="cr2fb_table">
<attribute name="cr2fb_jul23" alias="Jul23" aggregate ="sum" />
<attribute name="cr2fb_driver" alias="Driver" groupby="true"/>
<order alias="Driver" />
 <filter><condition attribute="cr2fb_scenario" operator="eq"
value="f6xxxxx0"/></filter>
</entity></fetch> HTTP/1.1
```

- Can use simple GET request, so long as length of URL does not exceed Dataverse limit, else use POST with GET in body of request
- Limited to aggregations that return less than 50,000 records (hence filters can be important)
- Limit to 5,000 records per request. But can use pagination to chain query responses to cycle through 5,000+

References:
https://learn.microsoft.com/en-us/power-apps/developer/data-platform/fetchxml/aggregate-data
https://learn.microsoft.com/en-us/power-apps/developer/data-platform/fetchxml/page-results?tabs=sdk

A digression – Querying Dataverse with TDS/SQL endpoint

```
SELECT COUNT([cr2fb_tableid]) FROM [cr2fb_table]
```

- Currently, only Power BI and SQL Server Management Studio can access (can’t access via Python)
- Fixed five (5) minute timeout
- The TDS endpoint can't be used with elastic (virtual) tables

https://learn.microsoft.com/en-us/power-apps/developer/data-platform/dataverse-sql-query

Fabric Lakehouse (1): Read a Lakehouse Table into Polars dataframe with partition

```
url=f'{self.Lakehouse}/Tables/{self.TableName}'
storage_options={"bearer_token": self.FabricAccessToken, "use_fabric_endpoint": "true"}
self.TableQuery=pl.read_delta(source=url,storage_options=storage_options,
                              pyarrow_options={"partitions": ParitionFilter}
                              ).lazy()
```

Polars dataframe

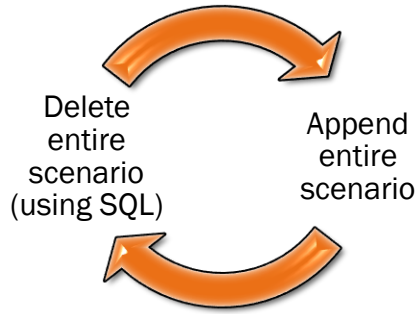
```
for Scenario in ListOfScenariosToLoad:
    PartitionFilter=[("Scenario", "=", Scenario)]
```

https://docs.pola.rs/py-polars/html/reference/api/polars.read_delta.html

Fabric Lakehouse (2): Writing a Polars dataframe to Lakehouse Table with partition

```
url=f'{self.Lakehouse}/Tables/{TableName}'
storage_options={"bearer_token": self.FabricAccessToken, "use_fabric_endpoint": "true"}
Table.write_delta(target=url,mode=mode,storage_options=storage_options
                                    , delta_write_options={"partition_by":ListOfPartitionColumns}
                                    )
```

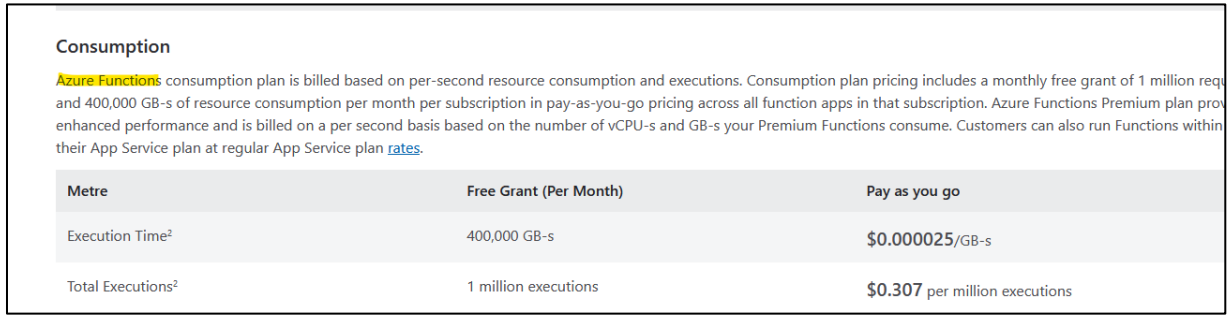
Mode options include “overwrite” and “append”



https://docs.pola.rs/py-polars/html/reference/api/polars.DataFrame.write_delta.html

Costs for my budget/forecast tool are likely to be low, with one key exception

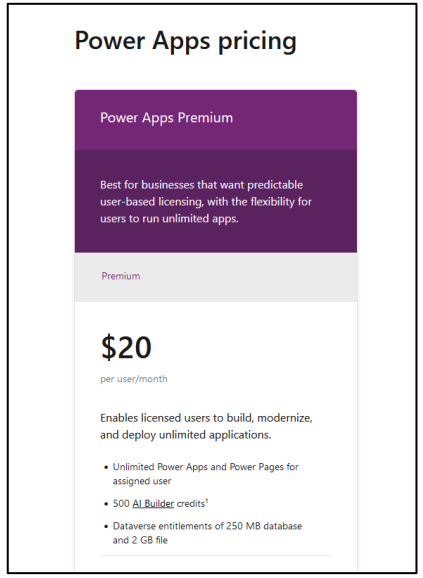
- Implementing the budget reporting in PowerBI rather than Power Apps reduces the number of Power Apps licenses required
- Azure Functions only incur charges when they are run, so naturally scale to use
- However …



Consumption

Azure Functions consumption plan is billed based on per-second resource consumption and executions. Consumption plan pricing includes a monthly free grant of 1 million requ and 400,000 GB-s of resource consumption per month per subscription in pay-as-you-go pricing across all function apps in that subscription. Azure Functions Premium plan prov enhanced performance and is billed on a per second basis based on the number of vCPU-s and GB-s your Premium Functions consume. Customers can also run Functions within their App Service plan at regular App Service plan rates.

Metre	Free Grant (Per Month)	Pay as you go
Execution Time[2]	400,000 GB-s	$0.000025/GB-s
Total Executions[2]	1 million executions	$0.307 per million executions



Power Apps pricing

Power Apps Premium

Best for businesses that want predictable user-based licensing, with the flexibility for users to run unlimited apps.

Premium

$20

per user/month

Enables licensed users to build, modernize, and deploy unlimited applications.

- Unlimited Power Apps and Power Pages for assigned user
- 500 AI Builder credits[1]
- Dataverse entitlements of 250 MB database and 2 GB file

Fabric would likely be the cost driver

SKU	Capacity unit (CU)	Pay-as-you-go
F 2	2	$0.36/hour
F 4	4	$0.72/hour
F 8	8	$1.44/hour
F 16	16	$2.88/hour
F 32	32	$5.76/hour
F 64	64	$11.52/hour
F 128	128	$23.04/hour
F 256	256	$46.08/hour
F 512	512	$92.16/hour
F 1024	1024	$184.32/hour
F 2048	2048	$368.64/hour

This is the trial capacity

To recap:

Agenda

- **Overview of the budgeting/forecasting project**
  - Motivation and requirements
  - Solution and architecture
- **Demonstration of the system**
- **Discussion of useful features**
  - Dataverse APIs and interfaces
  - Fabric Lakehouse and delta tables
  - Miscellaneous – editable tables, TDS endpoint, Dataverse/Fabric integration

In summary:

- Power Platform, Azure and Fabric are individually very capable systems that can be combined to create powerful business solutions
- The systems are constantly being updated, with the promise of even more functionality in the coming months and years
- Perhaps inevitably given the pace of change, some functionality is not well documented. Hopefully this presentation has shed light on some useful aspects.