Below are my comments from Friday’s questions.

I’d like to bucket my own questions for our Monday meeting into these topics:

1. What is the status of information from MGB?
   1. Is the table sent from Maggie correct? If so, is this the device summary data even though it looks like the raw HL7 messages?
   2. Will they be able to configure a mirror for testing at least a week before Sept 5th?
   3. Do they do hard deletes?
      1. Using timestamps will not work with hard deletes. I email BobK inquiring about using DSTIME on the 9th, which would work with hard deletes, but have not heard back from him yet.
2. What is the status from last week?
3. Clarification of terms
   1. I have created a document to provide a little history on the technologies that have led us to Iceberg.
   2. I think we need to stop using the terms “engine”, “compute engine” until we all agree on exactly what they mean and how they fit into this bigger picture. I expand on this under question “h”

a) beyond timestamps, are there other datatype conversion concerns or issues?

The current process converts IRIS datatypes into Pandas, Pandas into Arrow and then the PyIceberg library does the conversion into Iceberg. We still do need to make sure all mappings exist for this to be used as a general utility, but until we are 100% on all tables to be replicated a gap may still exist.

b) explore different storage tiers with ADLS2 (hot/cool/cold/archive).  any difference using them in this iceberg format?  Can we change the tier under existing data without moving it?  Can we do that while running?  Are there any facilities already for running mixed tier (the way our partitioned tables feature plans to do inside of IRIS)?  etc.

Good information to know about ADLS2, but likely not related to our specific development efforts.

c) what are the operational considerations/techniques with this config?  Standard monitoring? do we have to monitor errors and retries, or does the library do that already?

This was in the list of tasks for last week so will find out progress on Monday. The plan is having all error, warning, info and debug level logs in an IRIS table. Trace level logging not included in the table but in a local file.

d) is this actually interoperable?  Can we show that we can write and read and that other iceberg engines can read?  Do we think we need to lock anything for this kind of application?   (This is something Myles Collins could help with using his Snowlake setup; it's also something Boris Mamkin might use the spark engine for, since there's a rationale to learn about that engine even if we don't use it in what we recommend to MGB)

Since we are using the PyIceberg library, we can use it for reading as well. I’m not exactly sure what the locks applies to but there is no intention to lack any IRIS tables. Iceberg doesn’t use locks, except for access to the catalog, as far as I know. Writing is all done optimistically

e) (assuming we're using the IRIS-as-operations-container approach with embedded python).  What's the best way to move data in IRIS into and out of Iceberg format?  Can we easily make the data look 'native' to IRIS (ie if it's relational, the query pane in the SMP works, if it's an HL7 message it can look like a standard file to interoperability, etc)?  Does this 'just work' or do we have to wire up classes in python<->objectscript, etc.   That might benefit from a consult with folks that do embedded python a lot, like Guillaume

The current method being used to move data from IRIS into Iceberg is by reading the data into a Panda’s Dataframe, converting to an Arrow Dataframe and then writing this to Iceberg, which is handled by Pyiceberg. For simplicity I am using DB-API, but this will be much faster using over ODBC(setting up ODBC can be difficult so I opted for DB-API initially since other devs were joining). Making the data look native to IRIS requires writing code to expose it, probably as a Foreign table, which is required for any data not in IRIS. There is an important critical difference between Iceberg and most other databases, which stems from the fact that Iceberg is just a format, a specification. If this data was in Postgres, we could use SQL to interface, but we would still be using some XDBC driver to talk to the DB instance. For Iceberg there is no Instance, there is no database server. For this to just work, like via the SMP, the longer-term work being done by product needs to be in place.

f) what's the best way to get data from MDEV into an IRIS instance in Azure?  (I've been thinking mirroring as a simple-to-config, low-impact-to-production-ops approach, but it might be using interoperability and sending HL7 messages (if it turns out that that's what they need).

I’m not sure how a production interacts with a Mirror setup that is different than a non-production setup. We would not need the Interop messages, except for how they looked in the table and don’t think we want to replicate all of the Interop admin messages.

g) really rough performance gestalt (IRIS-on-Azure to storage, desktop IRIS->cloud IRIS)

This will almost entirely be independent of the code we write. It will be dependent on the read speed of the source IRIS instance and the write speed of the ADLS2 storage. Based on my work reading data from IRIS into Pandas dataframes and computing column level statistics, I would guess that 85% of the performance would be due to these factors and most of the rest would be serialization, pandas->arrow, arrow->iceberg. When interacting with Azure for Harris, we did find some severe network bottlenecks as well. The logs should capture the performance for easy analysis. We should run some sample tests on around 10GB to get some initial numbers(using ODBC instead of DB-API, and parallelization should be able to increase those by at least 10x based on my testing with stratification.

h) assuming we take the approach of making IRIS into a compute engine by using these libraries, what else do we have to consider besides the file operations and hosting the catalog??  (this is to Boris Mamkin's point about the embedded approach versus just standing up an engine)

We will not be making IRIS into a compute engine. We will not be building a compute engine at all. We should probably adopt the definition of compute engine to align with how Iceberg uses it(<https://iceberg.apache.org/multi-engine-support/>) as a reference to Hadoop processing libraries like Spark, Flink and Hive. From the Iceberg site:

**Apache Iceberg is an open table format for huge analytic datasets.** Iceberg adds tables to compute engines including Spark, Trino, PrestoDB, Flink, Hive and Impala using a high-performance table format that works just like a SQL table.

So our current implementation uses the PyIceberg library to copy data from IRIS tables to Iceberg tables. If we were to use a Spark engine/cluster, then the code we would have to write would be a spark job, written in either Java, Scala or Python and that job would ultimately run in the Scala Spark compute engine. In either case, we are not writing the code that is responsible for writing the Iceberg tables, that is the PyIceberg library. The Iceberg team maintains these implementations: https://github.com/apache?q=iceb&type=all