Data Persistence Project Group (Inaugural) Meeting – 8/21/18

Attendees: Jim, Chandra, Brandon (Dell), Michael (LF), Andre, Itamar (Redis Labs), Markus (ObjectBox), Rodney (Beechwoods), Steve, Andy (IoTech), Drasko (Mainflux). Attendees that may have joined after the start of the meeting may not have been captured and listed.

Discussion and action items as a result of meeting in RED

Old Business

- Requirements discussion - finalization
    - Current Must haves
        - License of product – compliant with Apache 2
        - Store and forward needs
        - Platform support:
            - Intel , ARM 64 bit
            - OS support: Linux, Unix, Windows, MacOS – those EdgeX has targeted
        - Support for batch writes is important
            - But this might require API changes
        - Performance
            - Need a holistic view. Small but slow is not acceptable.
            - Typically Prioritize writes over reads in performance
            - Concern impact of backup processes
            - General consensus: writes in the thousands of per second and in the range of kilobytes per milliseconds
        - Durable across EdgeX shutdown
        - Run in a container (Docker/Kubernetes/Snap/etc.)
            - Has to manageable from one control plane
            - Embedded might be considered by some, flexibility is more important to most (ability to easily swap out the database with proper app abstraction), and ability to distribute to alternate platform from micro services
        - Secure
            - Password protected
            - Supports data encryption (protect data at rest)
                - How we do key management? Already provided by Vault and we are exploring HW Root of Trust storage with security for Edinburgh release time fram.e
            - At least have the security functionality that we have in Mongo
        - Has Java, Go, C, C++ drivers/connectors
        - Community support and user-base size
            - Shows signs of significant use and contributors
            - Significant number of GitHub stars
        - Binary support (as long as we can identify what type of binary)
            - Max size (up 16MB)
        - Most use cases store data for day, a week not months
            - Capture quickly and ship it off
            - Provide enough data for historical back look for local actuation

            - Up to 10M data points on average; 100M max
            - Use case would dictate platform and architecture to deal with more or less
    - Nice to haves
        - Enterprise deployment support/support for commercial deployment
        - User administration / usability
            - Has ability to integrate to identity management
        - NoSQL (versus SQL) – assumed NoSQL given data types/objects; but immaterial otherwise. If resources allow, we may at least want to bench mark against one SQL database.
        - Synch capability
            - We must be careful not to exclude non-enterprise solutions with this nice to have.
            - This may be something the enterprise version offers.
        - Backup support
            - Again, careful not to limit to enterprise systems.
            - This may be something the enterprise version offers.
        - Transactional (ACID) or Eventual Consistent (which CAP axis)?
            - Availability and partition tolerance are priority
            - Totally use case dependent
        - Support multi-tenancy
            - Again use case/market driven
            - May not apply at the edge
        - 32 bit support (Intel or ARM)
        - Database that also runs in memory
    - **By way of no descending voices, the above is ratified as our loose requirement set**
- Evaluation
    - Current list of candidates
        - MongoDB (use as our baseline)
        - CouchDB
        - Couchbase (Couchbase Lite??)
        - ObjectBox
        - Key/Value DB
            - Redis (but support more data structures)
            - BoltDB (can run on a gateway; written in Go; lightweight)
        - At least on SQL (may be more useful for data like Metadata)
            - Postgres
            - MySQL
            - SQLite
            - CockroachDB – is this a cloud SQL database; can't run in single node config? Also written in Go; does not work natively on 32bit ARM
    - Can we get a volunteer person or team to build data access layer of Core Data against any one of these?
        - ***MongoDB (done with current implementation)***
        - Redis
        - CouchDB
        - Couchbase (Michael Hall – which version??)
        - ObjectBox

        - A SQL DB
        - Others above
    - Assuming we have a rough implementation of each, what tests do we want to make of each
        - How fast does Core Data with the database save an event with 10 readings? Speed test of writes
        - Determine maximum Event/Reading PUT throughputs. At what point does database not keep up with PUT calls? Load test on writes
        - How fast does Core Data with the database return an event with 10 readings by event id (presumably indexed)? Speed test of read
        - How fast does Core Data with the database return events by device name (presumably non-indexed)? Speed test of read
        - How fast does Core Data return 1000 events (and associated readings) given some query parameter (like where origin is between two timestamps)? Load test of read
        - Measure CPU and Memory usage during above tests
        - Test on multi-threaded , concurrent uses – simulating multiple clients each running tests above - concurrent load tests
            - Can we expand?
            - Identify/define level of load (light and heavy)
        - At what point does the database not keep up – what is the measure for that?
        - Look at how big the database is (how efficiently it stores the data)
        - How we run the tests
            - Test using the blackbox tests with some additions for scale and load
            - Run all the tests on the same box under the same conditions (LF or other). We want to provide for a level playing field.
            - Jim to work with Andy about performance platform/framework and how these efforts might help each other.
            - Jim to work with Trevor to outline where the DB touch points are – a guide to implementers of what work needs to be done for each DB.
        - How to deal with IDs (long term we don't want to surface IDs – but how to deal with IDs for the purpose of tests)
    - Action: Jim to add more detail as a strawman to these for next week's meeting.

    - Jim to lead paper exercise to compare on requirements list above and provide feature matrix (example, look at license, platform support, etc.)
        - Outline the matrix at next week's meeting
    - Volunteers for above work
        - Database consideration will require volunteers to help provide the implementation to test against.

New Business

- None