Rubin Observatory’s InfluxDB Enterprise Acceptance Criteria

Angelo Fausti, Frossie Economou

1 Introduction

At Rubin Observatory, we have used InfluxDB since July 2019, when we adopted InfluxDB OSS 1.x for the Engineering and Facilities Database (EFD). The InfluxDB, Chronograf, and Kapacitor stack has been very effective with our engineering data and has become an essential toolset for the observatory's operations.

In 2022, InfluxDB OSS 1.x reached end-of-life, and we initiated the migration to InfluxDB OSS 2.x. Today, our production environments at the Summit and the US Data Facility at SLAC run the 1.8.10 and 2.7.1 versions.

Both InfluxDB OSS versions, however, pose limitations, and we are contemplating the acquisition of an InfluxDB Enterprise license to ensure better support and stability for our production environments. Specifically, our interest lies in more granular backup/restore features, including support for incremental restores and the built-in High Availability (HA) feature offered exclusively by the Enterprise product.

This document provides an overview of the EFD and discusses our current challenges with the InfluxDB OSS product. Then, we present detailed tests for evaluating InfluxDB Enterprise to be executed during the Proof Of Concept (POC) phase offered by InfluxData.

2 EFD overview

Rubin Observatory's engineering data is collected at the Summit and recorded in a local InfluxDB instance. The observatory staff primarily uses the Summit environment for real-time monitoring of the observatory systems. The data is also replicated to the US Data Facility at SLAC through Kafka. The USDF environment serves the historical data to a larger user base and is critical for system integration, testing, commissioning activities, and later operations.

We expect a full data stream throughput of ~10 MB/s during operations. The Summit environment has a nominal retention period (RP) of 30 days, which amounts to ~5TB of storage, considering the compression in InfluxDB OSS 1.8.10.

At USDF, we have the requirement of storing historical data indefinitely. *We currently don't have a clear strategy for the long-term storage of the EFD data*. 1 year of EFD data represents ~50TB (see https://sqr-085.lsst.io for details). That may require the downsampling of high-frequency data and multi-tier storage to provide fast access to the most recent data (e.g., for the last 6 months or 1 year).

The EFD database is organized in shards with a shard duration of 7 days. The number of InfluxDB measurements in the database is currently around 1.6k, but the time series cardinality is low since we don't use tags.

2.1 Disaster recovery with InfluxDB OSS

Our main limitation with the InfluxDB OSS is the lack of support for incremental restores, which complicates the disaster recovery procedure and, in some cases, makes it impractical. See also this community post for a related problem.

Here, we describe the current backup/restore procedures for the Summit and USDF environments.

At the Summit environment, we run weekly backups of the EFD database and daily backups of the current shard. In this environment, we don't have file system-level snapshots in place. The database restore procedure relies on the InfluxDB OSS backup/restore tool and a set of InfluxDB Sink connectors (repairers) to sync deltas (recent data not present in the backups) from Kafka. A full restore takes approximately 48 hours. In a disaster recovery scenario, the summit backup of the EFD database must be restored to a secondary InfluxDB instance to minimize downtime of the Summit environment. The secondary InfluxDB instance eventually assumes the role of the primary instance once the repairer connectors finish syncing the deltas.

The increasing size of the EFD data at USDF makes the backup/restore procedure described above impractical. We recently started file system-level snapshots at

USDF, significantly reducing the restore time. However, we don't have a solution to restore data from outages longer than the Kafka retention period (72 hours) or from any deltas that cannot be restored from Kafka. The definitive solution to this problem requires the ability to perform incremental restores from backups to the existing EFD database in the live USDF InfluxDB instance - this is not currently possible with the InfluxDB OSS.

2.2 No built-in High Availability (HA) in InfluxDB OSS

InfluxDB OSS does not include built-in High Availability (HA). In a situation where we lose the InfluxDB volume, there's no other option than the disaster recovery procedure described in the previous section.

In particular, during database restores, the InfluxDB OSS instance often becomes unresponsive. In the HA setup, with at least two data nodes (or, equivalently, data pods in Kubernetes), we expect better performance in those situations.

2.3 Lack of support for Flux in InfluxDB OSS 1.8

InfluxDB OSS version 1.8.10 comes with an older version of Flux (v0.65.1).
While most EFD queries and dashboards are built in InfluxQL, Flux provides additional flexibility to create better dashboards and run complex analyses. That was one reason to migrate to InfluxDB OSS 2.x.

3 InfluxDB Enterprise Proof Of Concept

3.1 Enterprise entry-level package offering

- Cost $50,000/year
- 16 cores InfluxDB Enterprise License
- Platinum 24/7 Support
- 3 Single Node Developer Licenses
- 1 Non-Production Cluster
- Influx Insights and Influx Aware

3.2 Deployment Considerations

Our current production environments run on Kubernetes, and we expect to run the InfluxDB Enterprise Proof Of Concept on a similar Kubernetes cluster. In particular, using the InfluxDB Enterprise Helm chart instead of the InfluxDB OSS Helm chart.

Unless we encounter performance limitations, we will continue using our current storage solution based on WekaFS, a flash-based, POSIX-compatible, distributed, high-IOPS-capable network file system.

Based on the entry-level package offering, we'll deploy InfluxDB Enterprise in a High Availability (HA) setup with two data nodes (kubernetes pods) and 8 cores each.

To guarantee the InfluxDB data pods have enough resources in the node, we'll configure the CPU requests to 8 and memory requests to 96GB, respectively. We'll also add pod anti-affinity configuration to ensure the two InfluxDB data pods are not scheduled to the same k8s node.

3.3 InfluxDB Enterprise Evaluation

We plan on performing the following tests during the InfluxDB Enterprise POC phase:

- Use the InfluxDB Enterprise backup/restore tool to restore a ~5TB backup of the EFD database from our InfluxDB OSS backups to the InfluxDB Enterprise instance.
    - We expect the Enterprise backup/restore tool to be compatible with the InfluxDB OSS backup file format.
    - Example of command used to backup the EFD database using the InfluxDB OSS backup/restore tool:
      ```
      influxd backup -portable -database efd -rp autogen -host 127.0.0.1:8088 summit-efd-2023-08-10.influx
      ```

- Test the ability to restore a single shard (shard duration of 7 days) from an existing InfluxDB OSS backup to the live EFD database in the InfluxDB Enterprise instance and access read/write performance during the restore.
    - We expect the shard restore performance to be similar to the performance with the InfluxDB OSS restore tool or less than 4 hours.
    - We expect to be able to execute queries against the EFD database while the restore is in progress. We expect some read performance degradation but reads should not be significantly compromised. For example, a query execution time during the restores should not exceed twice the typical query execution time during normal operations.
    - We expect to write data to the EFD database while the restore is in progress. We expect some write performance degradation but writes should not be significantly compromised. For example, the Kafka InfluxDB Sink connectors should be able to continue writing to InfluxDB during a database restore. Specifically, the number of points per minute

written to InfluxDB should not drop more than half its value during normal operations.
    - These conditions ensure we can perform incremental restores without downtime of the EFD database using the InfluxDB Enterprise product.
- Evaluate the 8 cores x 2 data pods HA setup against a typical EFD data workload.
    - Compare query performance using the 8 cores x 2 data nodes HA setup with InfluxDB Enterprise and our current 16 cores InfluxDB OSS setup. We expect no significant query performance degradation to be noticed.
- Test InfluxDB Enterprise High Availability.
    - We expect to be able to execute queries and continue writing to the InfluxDB Enterprise instance after taking one of the data pods down.
    - We expect no data loss after taking one of the data pods down.
    - We expect a maximum 2x read/write performance degradation after losing one of the data pods.
    - This condition ensures no downtime in case of losing one of the InfluxDB Enterprise data pods.
- Test support to Flux in the InfluxDB Enterprise product.
    - We expect the InfluxDB Enterprise 1.10 or later to support a similar version of Flux as in the InfluxDB OSS 2.7 (>= 0.193.0).
    - We expect to run Flux queries through Chronograf and use the Python v2 client with the v2 compatibility API in the InfluxDB Enterprise 1.10 or later.

4. Migration path to InfluxDB 3.x

It's worth noting that InfluxDB Enterprise is rooted in the 1.x version line, and on-premise commercial support for the 2.x line is not offered. Drawing from previous conversations with InfluxData, a promising migration path for the future involves transitioning directly from InfluxDB Enterprise to InfluxDB 3.x Clustered, currently in development.