CEVAC Python Package Intellectual Disclosure

Bennett Meares, 22 July 2020

Table of Contents

Introduction

The CEVAC Python package is the foundational software library behind the CEVAC system, developed by Bennett Meares from May 2019 to July 2020. It orchestrates back-end processes like caching historical and live data and efficiently provides access to CEVAC data sets.

CEVAC Pipes Paradigm

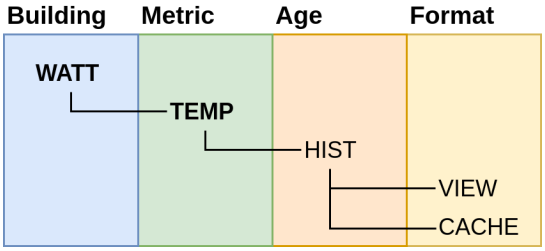

Figure 1: CEVAC Pipes contain collections of views identified as Ages.

CEVAC Pipes are collections of views, some of which are regularly materialized into on-disk cache tables. This design allows for high performance access to pre-aggregated data sets.

Every pipe must have a HIST_VIEW definition in order to be processed (see Solution 4 for more information).

Problems and Solutions

Problem 1: Large Data Volumes

The amount of available data is too large to handle at once.

Solution 1: Partitioning

Pipes partition data into a standard format: by building and metric (Figures 1, 2). This reduces overhead, increases availability, and makes analysis easier.

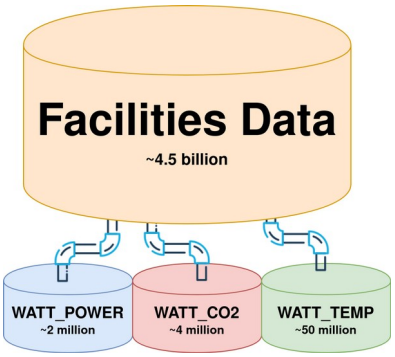

Figure 2: Pipes are fundamentally partitioned by building and metric.

Problem 2: Stress on Infrastructure

Large queries are stressful on critical Facilities infrastructure.

Solution 2: Caching System Scheduler

Pipes are cached by intermittently by the Caching System, thereby spreading the demand of large requests among many small, lightweight queries (Figure 3).

Figure 3: Pipes are cached by the caching scheduler, which balances the load among data sources.

Problem 3: Inconsistent Data Sources

Each data source has a different schema.

Solution 3: CEVAC Interface System

Standard data streams are defined by implementing the CEVAC Interface. Once a data source is applied to the interface, its Pipes inherit the standard Age aggregations and may be cached the same as any other Pipe.

The CEVAC Interface is used to generate Definitions for Pipes (Figure 4) as well as connect and update remote sources (such as pipes from Facilities' Oracle Servers).

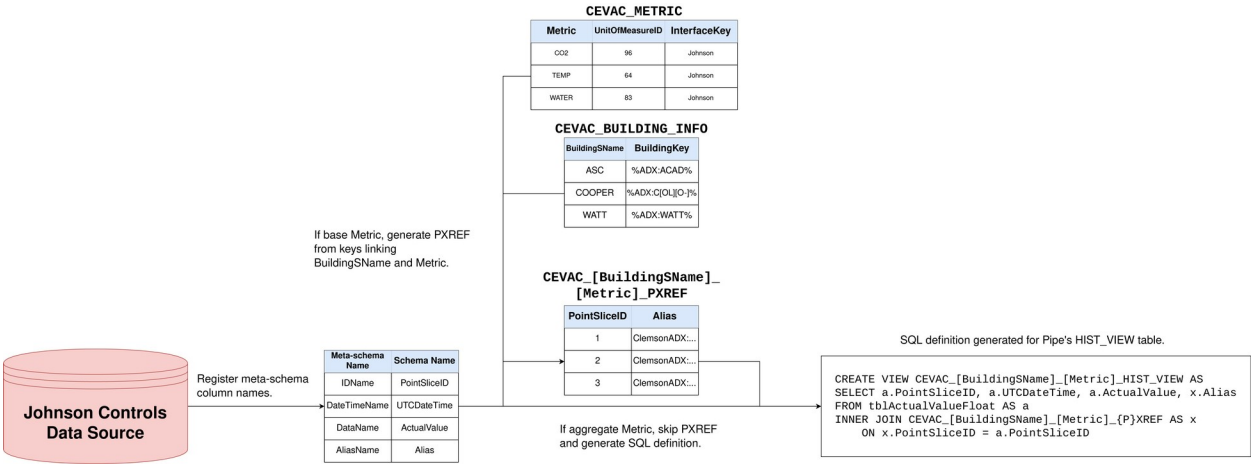

Figure 4: The CEVAC Interface normalizes data sources' schemata so that the CEVAC system can handle all Pipes, regardless of origin.

Problem 4: Raw Data Needs Aggregation

The raw data must be aggregated in order to see relationships and trends.

Solution 4: Custom Pipes

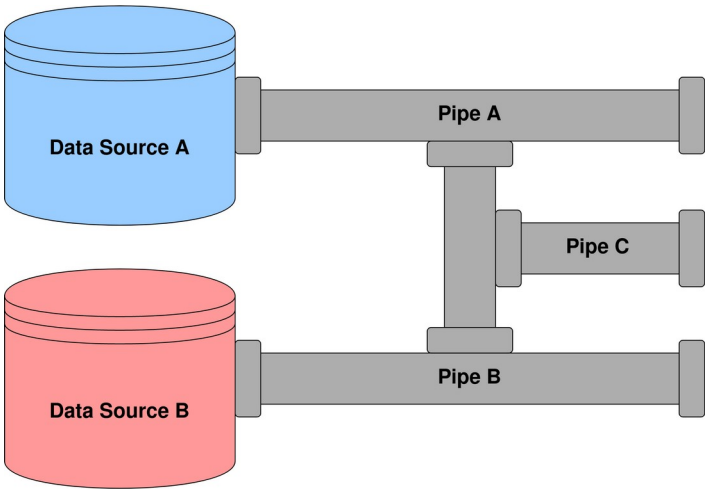

Figure 5: Pipes may source from other Pipes. In this example, Pipe C is derived from Pipes A and B, which each derive from separate data sources.

Custom definitions may be used for Pipes, and Pipes may be chained together to easily and efficiently create complex aggregations (Figure 5).

Consider the Pipe ALL_UTILITIES (Figure 6, Table 1). The definition aggregates raw meter data from Facilities and Schneider Electric into a comprehensive overview of each building's monthly usage, ranking, and carbon emissions.

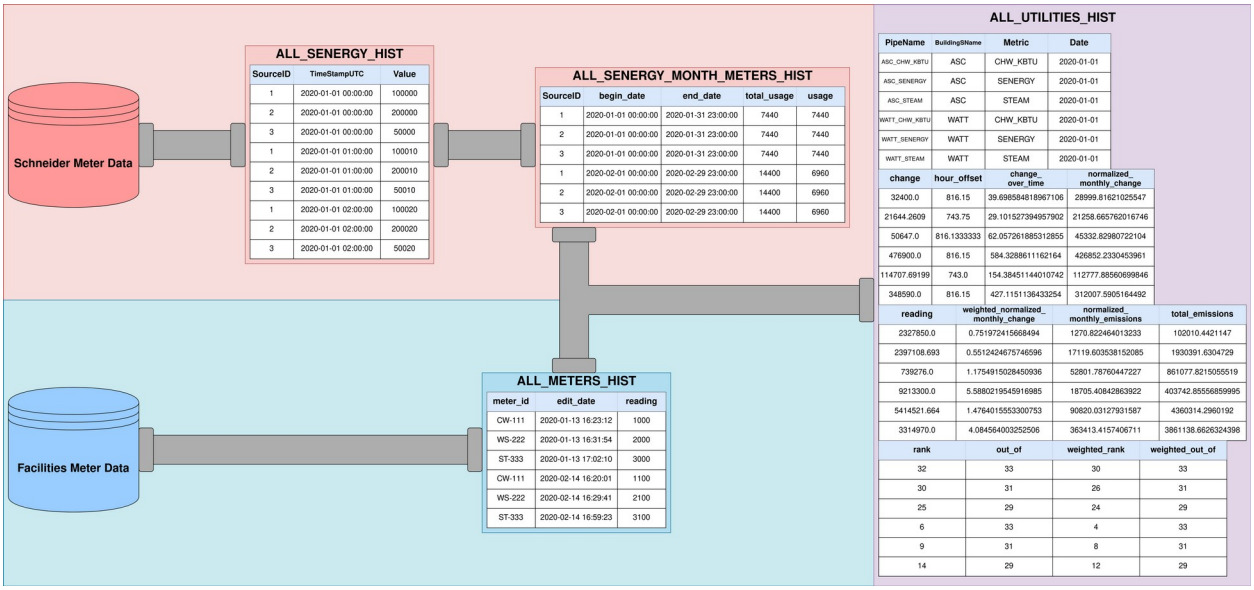

Figure 6: The Pipe ALL_UTILITIES complexly aggregates raw meter data from Schneider Electric and Facilities to calculate normalized and weighted monthly usages, emissions, and rankings for each building and metric.

Table 1: The many aggregations in the custom Pipe ALL_UTILITIES are derived from raw meter data.

Column	Description
`BuildingSName / Metric / PipeName`	Standard CEVAC building and metric identifiers
`Date`	Month and year (e.g. 2020-01-01)
`change`	Difference between this month's meters sum and the previous month's. Accounts for rollovers and resets.
`hour_offset`	Average number of hours between last month's meter readings and this month's.
`change_over_time`	Average hourly usage (`change / hour_offset`).
`normalized_monthly_change`	Average hourly usage multiplied by the average number of hours in a month (`change_over_time * 730.5`).
`reading`	Simulated master meter reading for the building (rolling sum of `change`).
`weighted_normalized_monthly_change`	The normalized monthly change weighted by the square footage of the building (`normalized_monthly_change / square_feet`).
`normalized_monthly_emissions`	Tons of CO_2 released per normalized month (`normalized_monthly_usage * EmissionScalar`).
`total_emissions`	Total tons of CO_2 released to date (`reading * EmissionScalar`).
`rank / weighted_rank`	Placement of a building's normalized monthly usage in comparison with other buildings (partitioned by `Metric` and `Date`). The value `weighted_rank` is weighted by square footage.
`out_of / weighted_out_of`	Number of buildings and metrics reporting for the current month and metric.

Problem 5: Administration

Pipes must be managed, maintained, and monitored.

Solution 5a: Actions

The CEVAC Package contains numerous Actions which may be applied to Pipes. Actions share a standard set of arguments and may be triggered via the command-line interface.

For example, the command

```
python -m CEVAC -x update_cache
```

translates to “execute the update_cache method for all existing Pipes”.

Pipes may be explicitly listed with `-b` and `-m` flags and filtered with the `--params` flag. Consider the table below containing several example commands and their explanations (Table 2).

Table 2: The CEVAC Python Package uses a versatile argument system to manage Pipes.

Command	Description
`python -m CEVAC -x bootstrap -b ASC,WATT -m WAP`	Bootstrap the Pipes ASC_WAP and WATT_WAP.
`python -m CEVAC -x live --params isProduction:1`	Run the live data scheduler for all production Pipes.

Solution 5b: Administrative Console

The CEVAC Administrative Console provides a graphical interface for managing Pipes. The Console translates the user’s actions into the standard arguments defined above.

For example, in the pictured screenshot (Figure 7), clicking “Build Pipe” translates to the arguments

```
-x bootstrap -b WATT -m CO2
```

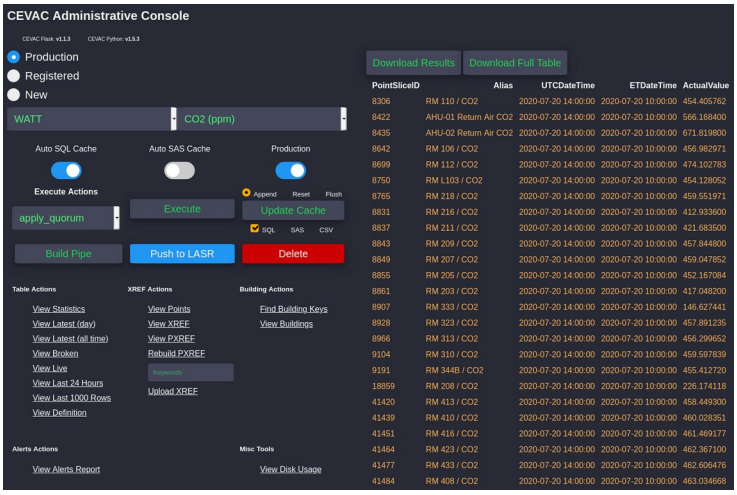

Figure 7: The CEVAC Administrative Console allows users to easily interact with the CEVAC Python Package.

Problem 6: Providing Pipes To Developers

Developers need to access CEVAC data to build applications.

Solution 6a: Pipe Objects

Developers use the CEVAC Python Package to access data through Pipe objects. The package includes a `getPipes()` method for constructing Pipe objects. Referencing the `.data` member of a Pipe's Age fetches and maintains the table's data. Consider the code snippet below:

```
>>> import CEVAC
>>> pipes = CEVAC.getPipes()
>>> pipes['WATT']['SENERGY']['HIST'].data
```

Pipe objects fetch tables on demand and store them as pandas DataFrames (a useful format for data analysis) and index DataFrames by ID and date-time columns. These DataFrames may be stored in-memory or on-disk.

Pipe objects are designed to sync with the CEVAC database to efficiently provide the most up-to-date data on demand. Pipes may be configured to sync data from the CEVAC REST API (see Solution 6b) or directly from the CEVAC SQL Server.

Solution 6b: CEVAC REST API

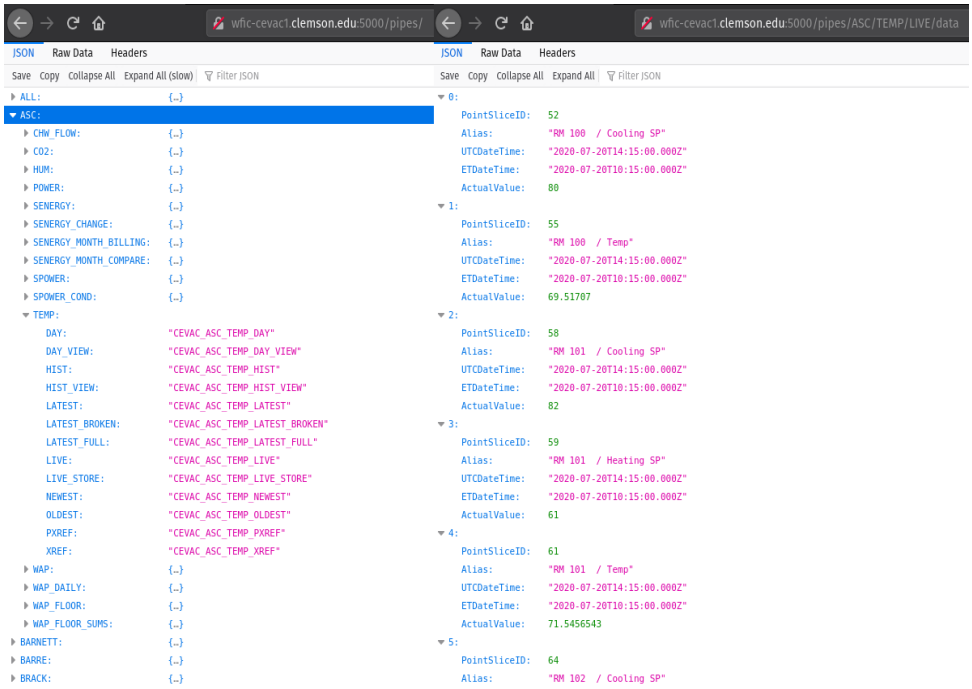

Figure 8: The CEVAC REST API provides easy access to CEVAC Pipes.

The CEVAC API is a RESTful API for serving CEVAC data and is analogous to the functionality of `getPipes()` (Figure 8).

The API is designed to be scaled to meet high demand and preserves memory by caching Pipes on disk (Figure 11).

Consider the table below which demonstrates the performance improvements of using the CEVAC API over directly accessing the CEVAC database (Table 3, Figure 10).

Table 3: The CEVAC API efficiently serves JSON data.

Test	Time
CEVAC API (warm cache)	13.950 seconds
CEVAC API (cold cache)	42.392 seconds
Direct SQL Query	56.185 seconds

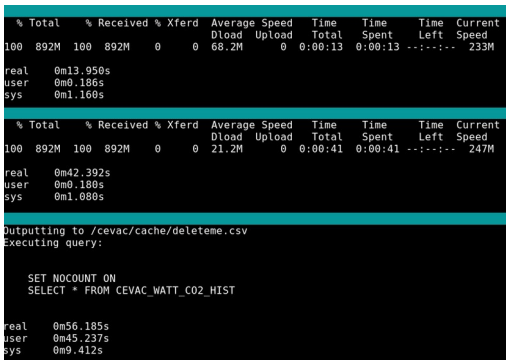

Figure 10: Screenshot of the figures presented in Table 3.

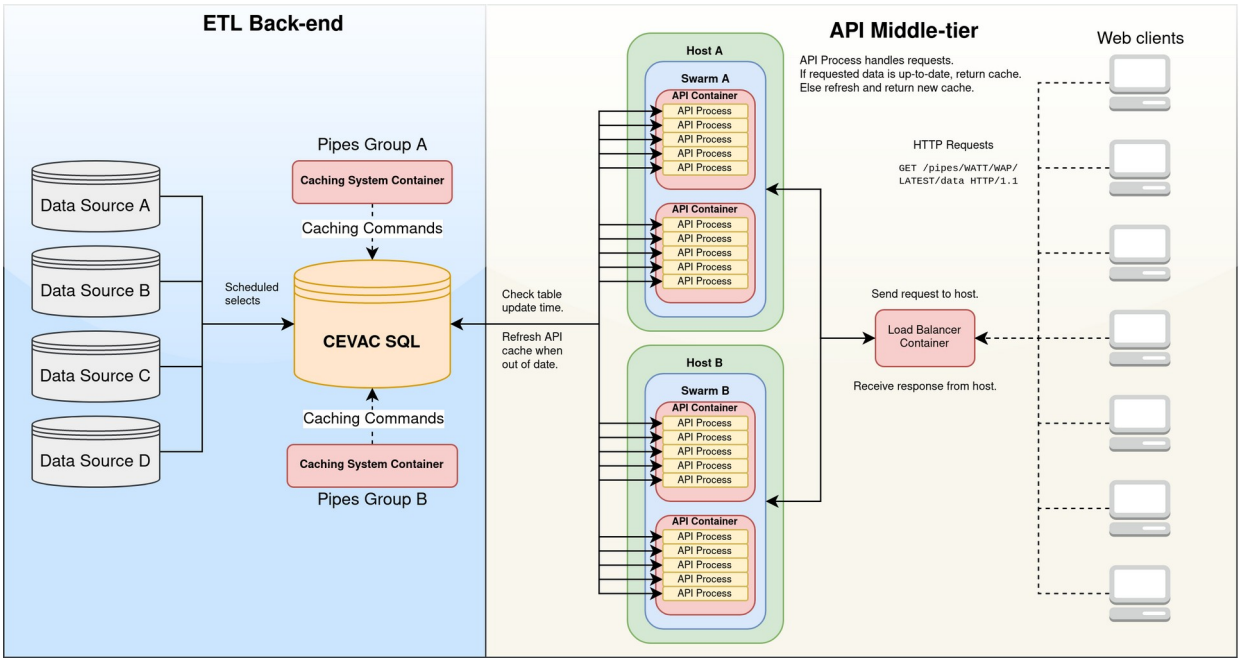

Figure 11: The CEVAC REST API protects the SQL Server by leveraging CEVAC Python to cache Pipes.

Problem 7: Live Data

Customers want to know the real-time status of certain Pipes on-demand. However, Johnson Controls Pipes update irregularly, and the caching system takes approximately 15 minutes to update all Pipes. Therefore data may not be available for several hours (until the Johnson Controls data source contains the latest historical data).

Solution 7: Live System Scheduler

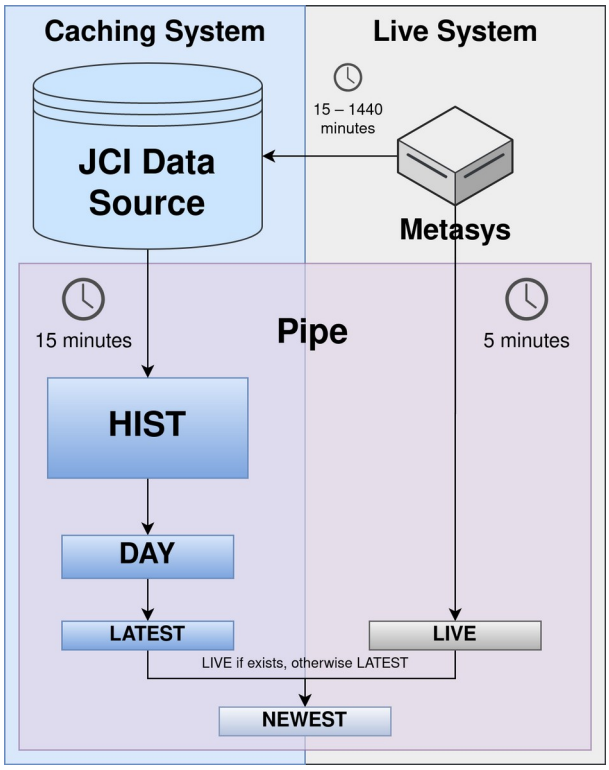

Figure 12: The Caching System updates Pipes' HIST tables, and the Live System creates LIVE tables from the JCI Metasys API.

Johnson Controls Pipes contain the Age LIVE, which is updated every 5 minutes directly from the JCI Metasys API. LIVE tables are created by the Live System (Figure 12), which emits live data via MQTT, a publish-subscribe protocol (Figure 13), and stores live data for 24 hours, reducing the frequency of older stored live data to 1 hour (Figure 14).

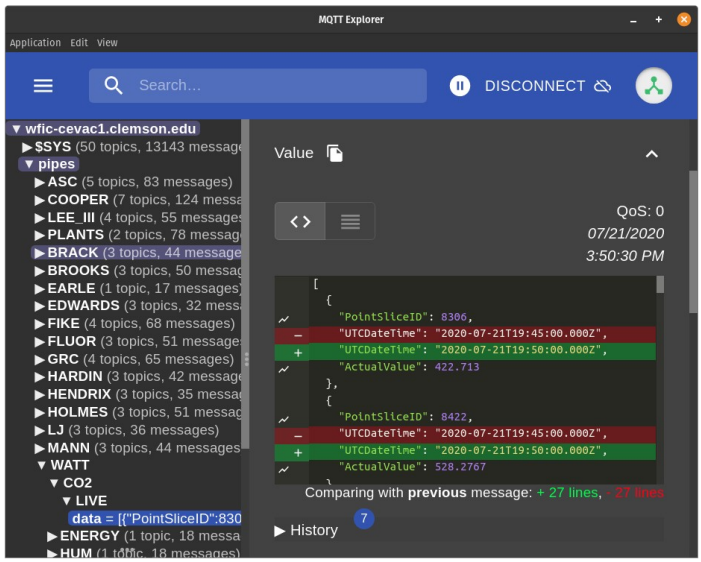

Figure 13: Clients can subscribe to live data through WebSockets using MQTT.

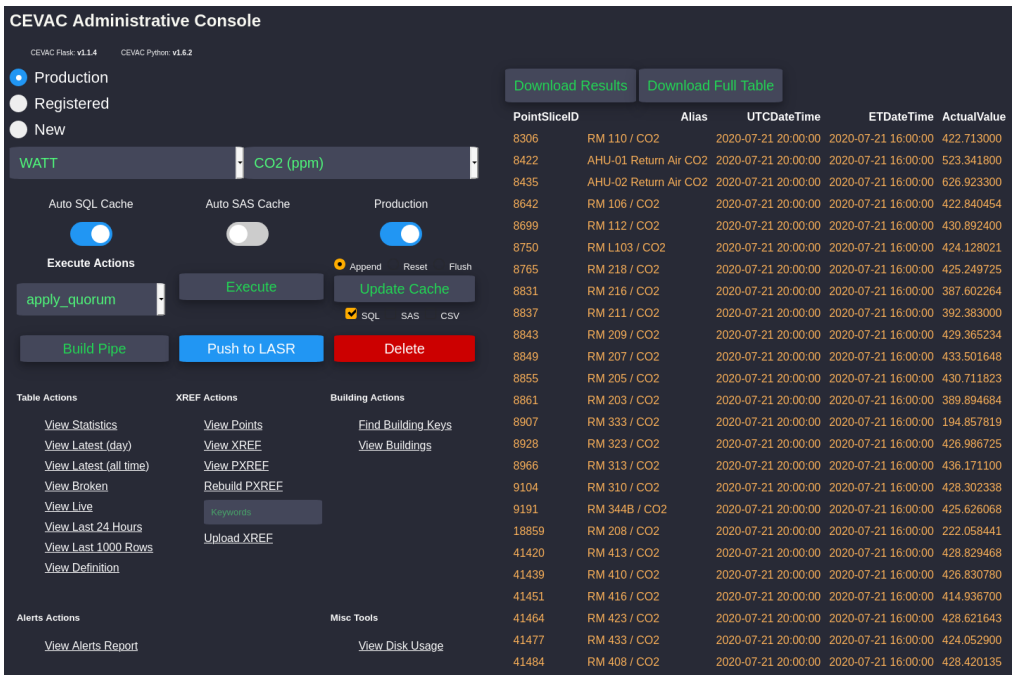

Figure 14: An example of LIVE data for the Pipe WATT_CO2.