**How would I build the architecture for the larger datasets?**

My assumptions:

* We are receiving temperature reading from weather station sensors every second in real-time from all the cities of Canada.
* We are looking for a way to ingest the sensor data in real time.
* We run the ETL job at the end of the day to process all the data that we ingested.
* Run analytics on the big data.

1. Data Ingestion:

As now we are ingesting data from Canada cities every second, the number of daily temperature is going to be:

(Number of cities in Canada) \* 86400

= 1000 \* 86400

= 86 million records daily

As the temperature reading comes from the sensor in real-time, I would probably use a pub-sub layer between the source and the data storage. The pub-sub would ensure the asynchronous consumption of the data by the subscriber. Also I read some article where we can use tools like Apache beam to do some data transformations for the chunks of data in Pub-sub before loading the data into storage. Well this could be important for operations like finding median of the temperature in real time. Let’s say we want to find the median of the temperature every hour in real time. Then we first need to sort the data and calculate the mid value from the data chunks in pub-sub storage and then store the result in some database while loading the temperature data in some kind of HDFS storage.

1. Data storage: Then I would probably store the data in the distributed environment like HDFS. We don’t want to store every second data into a HDFS file, as this will lead into multiple small files. Rather we would consume data from the pub-sub after few hours.
2. Data processing: Now that we have the temperature reading data into our HDFS, we need to perform some ETL operations so that we can drive some meaning out of the data. We could use tools which supports distributed data processing like Apache Spark which supports in-memory computation. I am thinking of going with ELT approach. Firstly, we get the new data from Hadoop and use tools like Apache Spark to load the data into some database. The database of my choice is Apache Cassandra. I am wondering the partition key that I would select for this database. As I did some research around 100 MB should be a good size for a partition. As 86 millions records is a huge number, and is likely to exceed 100 MB, I can’t select date as the partition key. What if we decide to go with hour as a partition key and we have 8 nodes? Then we would store around 3.5 million records in each partition and a node will contain 3 partitions. Also important is the clustering key. It defines the order on how we want o physical store the data on disk. The data should should be sorted by the latest record first.

Next, we run an incremental job every day probably at night time which extracts the data from the Cassandra using the partition key and clustering key. As we can perform range operations using clustering key, the pseudo-code to get the daily data could be something like:

Df <- Empty dataframe

For hour in range (1,25,1):

Query = “

SELECT \* FROM table

WHERE partition\_key = {hour} And clustering\_key >= todays\_start\_time\_stamp AND clustering\_key <= todays\_end\_time\_stamp”

Df.append(get\_data\_from\_cassandra(Query))

Then we can do the transformations and store the data for analytical purpose. We can go with databases like MongoDB or ElasticSearch for the analysis layer. Again, the schema design would depend on the type of read query and the number of nodes would depend on the number of read requests. For number of nodes, we can set the options to auto-scale for some of the environments (Eg: Mongodb in atlas).

1. To retrieve the data from the analytical database, assuming we have a high read request, I would use REST APIs that are async in nature. So, I would probably go with Node JS. However, we could also use Python based FastAPI(which is fast as the name says.) FastAPI is very popular these days for deploying the machine learning models.