# VergeSense IOT Initiative

**26th December 2019**

## OVERVIEW

Design a system that can handle device pinging and tightly batched requests to write sensor data. The system should also be able to respond to read requests of the data. The system should be preformant and efficient

## BUSINESS CONSTRAINTS

1. During brewing, our devices post live time series data about the state of the coffee every 0.5 seconds over the span of 4 minutes totaling 0.5mb of data
2. When not brewing, our devices ping the servers once an hour with 1kb of data
3. While traffic is always hitting our servers, we see significant spikes between 7-9am and 1-3pm
4. Presently, customers can only query by 3 time intervals (last 7 days, last 30 days, last 365 days) and when they do, CPU on the server spikes due to the ad-hoc computations that we run on the historical data

## IDEATION and SPECIFICATIONS

It sounds like we have a couple of considerations when we are designing this system: On the write side there are two kinds of behaviors,

1) ping data which happens at a regular interval and

2) brew upload data which will happen en masse for a short period of time.

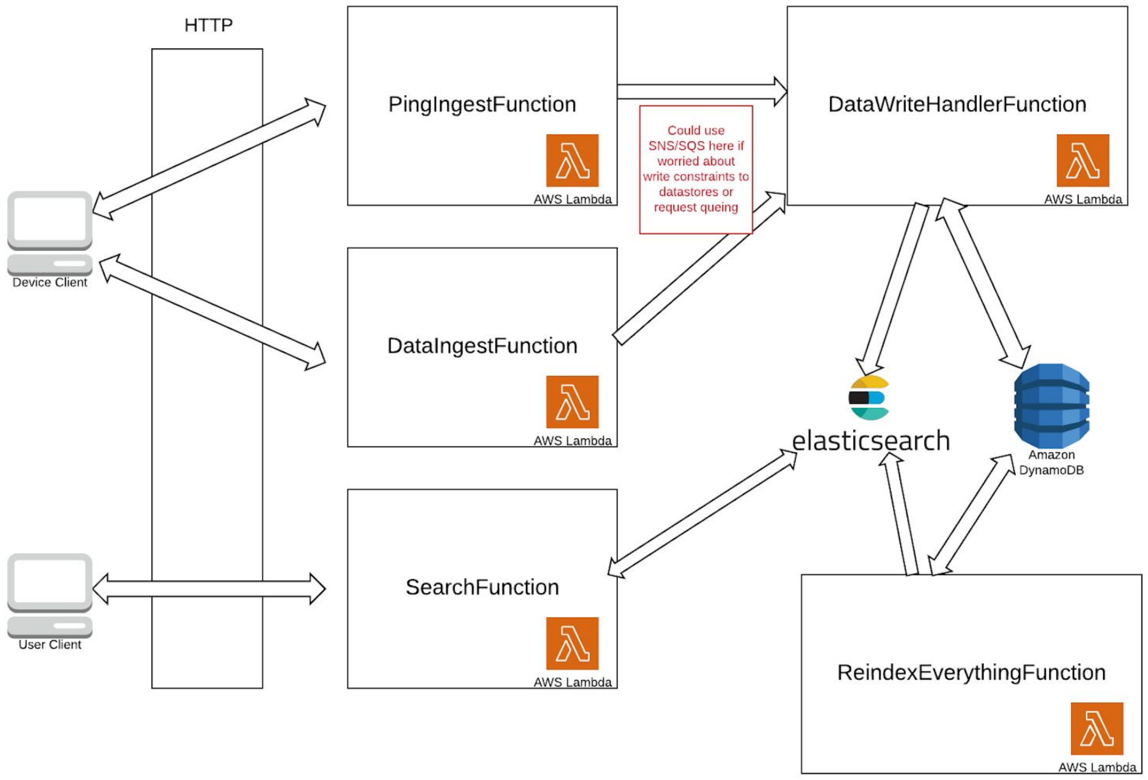
On the read side it sounds like the requests will be staggered with some requests asking to read a lot of data and some requests asking to read a smaller amount of data.

On the write side I would break the two use case up to two distinct problems and use a function-as-a-service approach to handling the load. I would use something like AWS Lambda to handle the writes and split the functions in two so that one function would handle the "ping" data coming in write to the database and the other functions would handle the"spikier" brew data coming in and handle that. In both of these services I would then write the data to primary datastore. In this case AWS Dynamo DB seems appropriate as you could index the data based on time series. In addition to writing to the primary data store I would also write to a secondary data store which is read optimized and would be used to handle the read/search interface. I would use a datastore like Elasticsearch as the secondary datastore. I would choose this because I have worked with it in the past and can be hosted in AWS or with hosted third parties and that easily handles any sharding and any failover precations that would need to happen. On the read side I would then make an HTTP interface which would be another function-as-a-service approach which would most likely leverage AWS Lambda to handle incoming HTTP requests and then query the Elasticsearch cluster and return the data that the user requested (whether it be 7 days, 30 days or 365 days). I would not be concerned with either elasticsearch or lambda handling the queries as they should be read optimized. The things I would be concerned about would be the size of the data being sent back and the client's ability to render the data if it is a lot. I have run in to situations before where the UI is the bottleneck in displaying large sets of data. Something to be mindful of.

One thing I have not touched on is that if there are two datastores (Dynamo and Elasticsearch) how do we keep those datastores in sync? I would propose that there is a process which is able to take in raw sensor data and write it to both data stores. Additionally I would propose there would be a way in which we could take all the data that is in dynamo delete all the data that is in elasticsearch and write all new data from dynamo to elasticsearch. This process would take a long time but it would allow us to always recover and write data from the primary datastore to a read optimized datastore.

Another thing that was not addressed is that there may be some back up at the HTTP layer when we are writing large amounts of data en masse. One way we could mitigate this issue would be to just take anything that is written to the function-as-a-service and immediately placing it on an SQS queue to be processed later in the backend. This may be over optimizing and I am not sure that it would be needed especially with the performance characteristics of AWS Dynamo but if we ever got in a situation where we could not efficiently write to the data stores then we could put a background worker/queuing mechanism in place to ensure we could throttle the writes when needed.

## DIAGRAMS


HTTP
PingIngestFunction
AWS Lambda
DataWriteHandlerFunction
AWS Lambda
Could use SNS/SQS here if worried about write constraints to datastores or request queing
Device Client
DataIngestFunction
AWS Lambda
elasticsearch
Amazon DynamoDB
SearchFunction
AWS Lambda
User Client
ReindexEverythingFunction
AWS Lambda