**Butterwire Backend Python Engine Test**

Overview:

This test is not about extensive work – I expect this to take around an hour or so.

The goal is to demonstrate use of Python and appropriate libraries. As well as an ability to quickly get hold of cloud based services and demonstrate of your experience.

The Test:

This is a simple Ingest – Process – Export flow (similar to ETL).

Have asked for financial data, but happy to use any numerical data. Feel free to use any web service or DB technology (we use Postgres and Dynamo currently) if that is more familiar.

Libraries core to butterwire:

AWS Boto3

NumPy for process calculation

Pandas for data manipulation

SciPy, SqlAlchemy

This doesn’t have to be too pretty – but keen to see your architectural decisions, very interested around your thoughts and examples of testing, scale and validation of data coming from external web-services.

How would you deal with sparse data and appropriate error handling – i.e. just throwing errors is not always suitable.

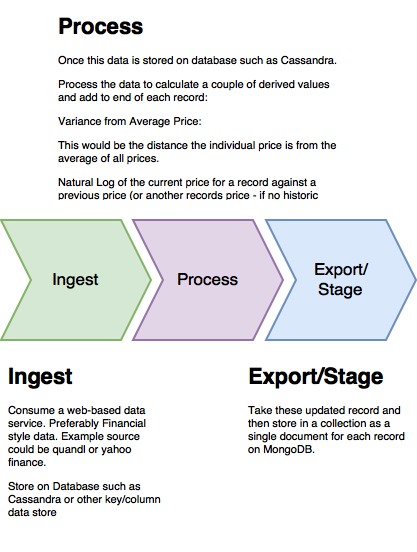
Finally – the goal is to host all of this on cloud infra – although this is not necessary for the test, consideration of this in your solution would be good.

Summary:

Quick and Dirty – but indication of what you would do to make it more industrial.

We will use your code as a discussion point in the interview. Butterwire is moving from proof of concept to commercial release. You will be instrumental in helping us get there.

NOTE: Target output doesn’t need to be Mongo – but JSON is our format of choice.



## **Discussion**

The formulation of the exercise is a bit confuseing, which is probably one of the goals of the exercise.

It raises some questions that need to be addressed i) on the formulation of the request ii) on the architecture and, iii) on the deliverables.

### **i) Formulation**

* The source of the data and the domain context are loosely defined, there is no specific indication . *Here we assume the process is reading a timeseries form external service, quandl, and the time series of interest is 'LME/PR\_CO', as it is available for free.*
* Is data staged in "cassandra" database and piped in memory to next stage or processing stage reads from database? Why this decision? i.e. what volume of data are we processing? *Here, we assume data is staged in cassandra and data structure is passed in-memory. Moreover, the DAO for Cassandra is mocked and injected so that we can cut dependencies on a proper installation. Prod environment requires to configure and inject a proper connector.*
* Variance has a precise statistical definition, I guess they are trying to redefine it. However, the information provided is insufficient, as rigorous definition for "distance" is missing. A more precise definition of this measure would be helpful. *Here, we assume the request to de-mean the time-series.*
* The 'log of the price of a record against a previous price' is poorly defined, a parenthesis is missing at the end of the sentence, probably part of the definition is also missing as a generic 'previous value' makes poor sense, 'against' is not a mathematical operator, a proper operator e.g. the ratio or the squared difference are much better formulations. Adequate answer is domain specific, Having adequate context could help operating decisions. The logarithm is defined for values bigger than zero, so 'against' should produce strictly positive output. *Here we assume the request to estimate the log-return defined as lr = log(v2/v1) = lo(v2) – log(v1)*
* 'another records price if no historic' is not rigorous. A better formulation can be a 'default' initial value or a precise indication of what specific record to use in each case, for reproducibility. Both suggestions are strongly dependent on the domain. *Here we use a parameter to allow the caller to pass a default value, which is set to default in the main.*

### **ii) Architecture**

The proposed architecture somehow is confusing. Some information about the process is missing, e.g. the volume of data. However, a few questions arise.

What is the role of cassandra? Is it used to stage data just in case, while data is passed in memory to processing stage or it is used as interface between ingest and process (i.e. ingest is persisting to cassandra and process is reading from cassandra), the former case might be adequate for relatively small volumes of data manageable in memory, the latter case, twinned with token aware map-reduce facilities is adequate for high volumes of data that do not fit in memory and require to scale out. The current formulation appears to mix the two cases taking a fairly inconvenient position between having burden of managing a big-data database and dealing with BASE vs ACID consistency, while processing small data that an better fit in a properly sized SQL database (e.g. MySql or PostgreSql).

Use of mongoDB saving a single data point per document might be required for some specific boundary condition, a proper evaluation of performance would require profiling the process; however, if the data needs to be read as a time-series, which is quite likely to happen considering the nature of the application, having each time point in a separate document in MongoDB might not be the most efficient way of storing data.[[1]](#footnote-2)

### **iii) Deliverables**

The problem is very rich and it touches a number of aspects of a fully featured processing pipeline. Considering the number of components involved and the need to set up a non trivial processing pipeline, the number of architectural decisions involved and the nature of the problem, a full implementation is clearly way above the scope of an interview test.

Therefore I suggest to reduce the scope of the deliverables to something reasonably achievable in half day.

Unless directed otherwise deliverables should focus on

* Ingestion module connect to external Quandl service, with example of back-off policies and critical failure handling. Injection of Mock DAO to insulate dependencies from Cassandra (one can inject a proper DAO depending on the database choice any time). Data are returned as in-memory pandas DataFrame.
* Processing module: under assumptions in I), the module is receiving input data through the orchestration, performing computations on the data and returning an in-memory pandas DataFrame for export.
* Export/Stage: the module is receiving input data and marshalling the DataFrame to json row by row. Optionally sending to a mock connector for MongoDB, and returning the staged data.
* An example of python orchestration module. For real production one should consider using a framework of appropriate power/complexity or use facilities provided by AWS.
* Unit test
* Example of configurations with templates e.g. different parameter values for different environments (dev, ci, test, prod environments)
* Documentation

## **Instructions**

### Structure of the package

/butterwire: the implementation of required modules

/docs: documentation, both a copy of this file and Sphinx

/tests: a set of unit tests

### **Configuration**

The root directory contains the configuration template config.ini.template, please copy it to config.ini and set the properties to the proper values, e.g. quandl\_key is required and it needs to be replaced with your key (in case I can provide mine for test purposes).

The main uses the default configuration. It can be overridden setting the environment variable to a different value before running e.g. on posix systems

export ENV=TEST

### **Makefile**

The makefile has 3 options:

init: uses pip to load dependencies

freeze: freezes dependencies and overwrite previous file (if run in error, please restart form the original installation)

test: run the test to verify the software works in the current setup

### **Preparation and Target Environment**

The target machine requires an installation of python3 (tested with 3.5.2), pip, make, and venv. Additional tools might be required to compile Sphinx documentation.

Before starting it is recommended to create a virtual environment for the project to avoid interfering with the local installation and activate it[[2]](#footnote-3).

### **Initialization of the Environment**

To initialize the environment run

make init

The Makefile uses pip to install the dependencies specified in requirements.xtx

### Unit Test

Once the dependencies are installed one can run the test using

make test

nosetest run 6 unit tests on the modules in butterwire.

### Example of Usage

The file main.py contains an example of usage

The file configures logging, configures and instantiates ingestor, processor, exporter, and orchestrator, injects dependencies and runs a single IPE pipeline on predefined data. The example can be extended to incorporate the orchestration in a more comprehensive application.

1. https://www.mongodb.com/blog/post/schema-design-for-time-series-data-in-mongodb [↑](#footnote-ref-2)
2. See e.g. http://docs.python-guide.org/en/latest/dev/virtualenvs/ [↑](#footnote-ref-3)