Large json ingest and aggregation in ES

Example: daily inventory updates

Problem Statement

ACME uses elastic search to support interactive queries on their inventory receipts. The way this works is below.
Every day ACME gets a file (array of JSON objects) that contain items received globally across all their warehouses.
Each JSON object has a “key” (UUID) that uniquely identifies the inventory type.
An example of one such document is:

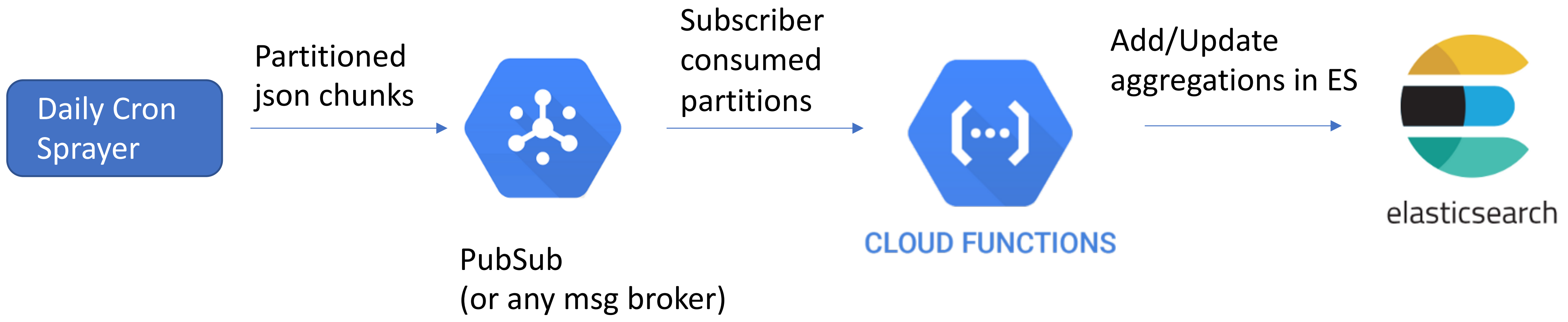
```
{
  “key” : “9f7bba2f-51df-48fc-9da4-9bf5da67b152”,
  “name” : “AC Compressor Gasket”,
  “quantity” : 7153
}
```

On a typical day, the file contains about 200M records - each record representing an inventory type identified by a unique key.
To support query of this data, ACME ingests that data into elastic search with the following additions. It adds three fields to create an elastic index that looks like:

```
{
  “key” : “9f7bba2f-51df-48fc-9da4-9bf5da67b152",
  “name” : “AC Compressor Gasket”,
  “quantity” : 512345,
  “first_received” : “2019-07-09”,
  “number_of_days_received” : 38,
  “last_received” : “2020-09-17"
}
```

Here the first_received field represents the first time an inventory item type is seen in the input file.
The number_of_days field represents a number that increases every time the input file has the specific inventory type.
The last_received field represents the latest date when the inventory type was seen in the input file (received in its warehouse).
Write a program that is meant to run daily to create/update this elastic search index.
This program will create new documents in the index, if previous ones do not exist. It updates the existing document’s “name” field if the input file has a new name for the same key, increments the “quantity” field based on the “quantity” from the input field.
If a given key exists in the input file for that day, It increments the number_of_days_received field and updates the last_received field.

PoC architecture in GCP and performance



Chunk size: 1k/msg
Spray: 3k/sec
(200M -> almost 1 day)

Throughput: 1k/3sec
Parallelism latency depend on scale out workers and elastic cluster write performance

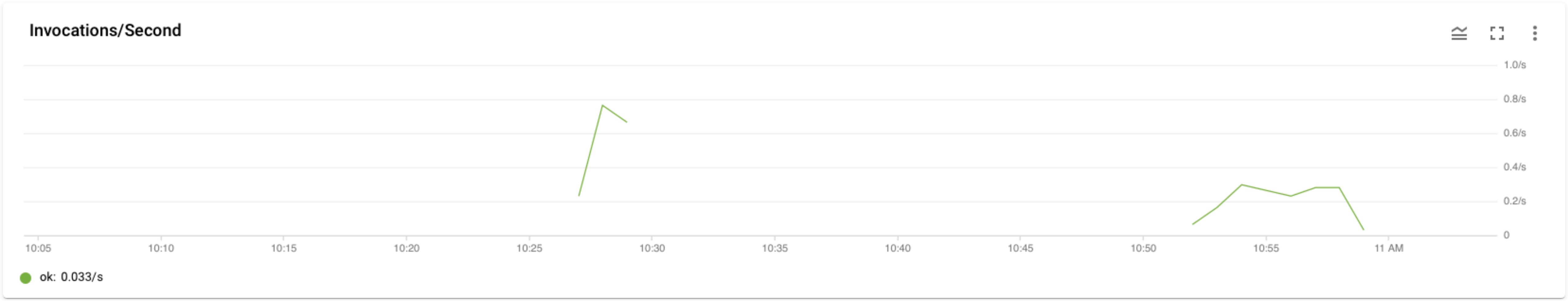
Scale	Cloud function invocations	Longest pole
10k	10 (latency 15 ~30 sec) – 1k/msg (128MB RAM)	30 sec
100k	100 (latency 11~45 sec) – 1k/msg (128MB RAM)	18:26:30~18:29:00 (150 sec)
100k	10 (latency 11~45 sec) – 10k/msg (256MB RAM)	18:32:00~18:34:30 (400 sec)
1M	100 (latency 11~45 sec) – 10k/msg (256MB RAM)	18:51:30~18:58:10 (400 sec)
100M	10000 (latency 11~45 sec) – 10k/msg (256MB RAM)	Estimation ~ 100x400sec (~11hrs)

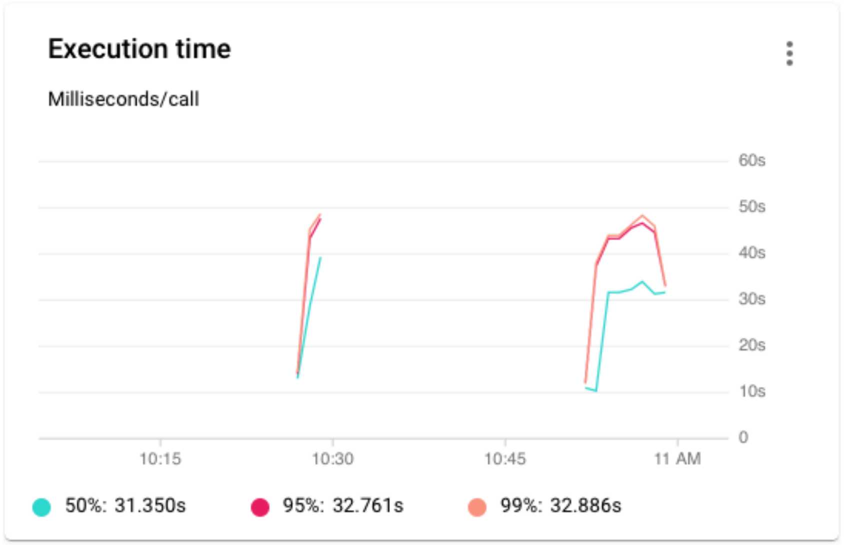
10x scale with ~2.6x latency. Assuming scale to 200M w/o higher batch level and scale linearly - so 200M can complete daily

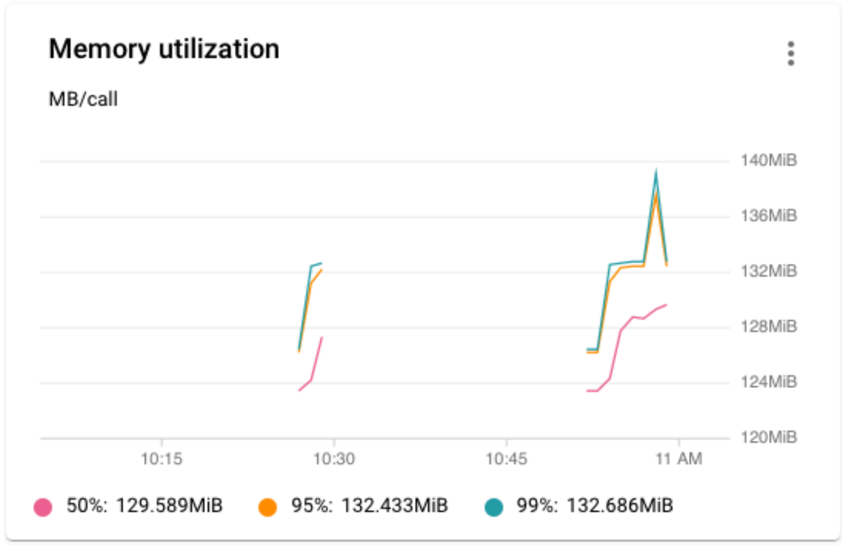
GCP daily cost – ingest and aggregation 200M rec

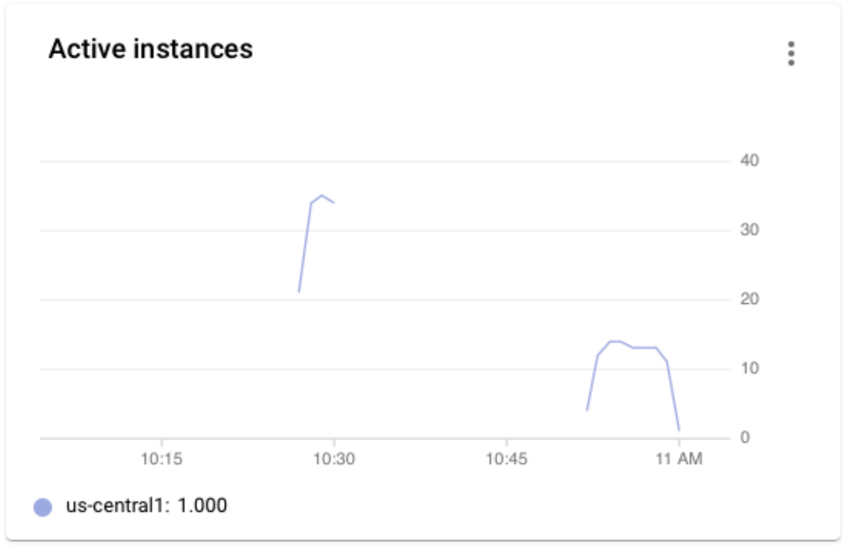
Services	Unit cost	Total
PubSub	$40/TB, (100 bytes/rec) * 200M = 20GB/day	$40/(1000/20) = $0.8/day
Cloud Function	$0.0000004/invocation, 1k rec/msg invocation	$0.0000004 * (200M/1k) = $0.08/day
	$0.000000231/0.1sec, 3sec/msg	$0.000000231*(200M/1k)*3/0.1 = $1.386/day
Elasticsearch (VM)	Assuming 200M rec in mem indexing (100b/rec) 20GB data with 2x indexing	$0.536048 * 24 = $12.86 (e2-standard-16)
Elasticsearch (Managed)	TBD	

Cloud Function stats – 100k vs. 1M records ingest (better strategy with higher bulk&batch)



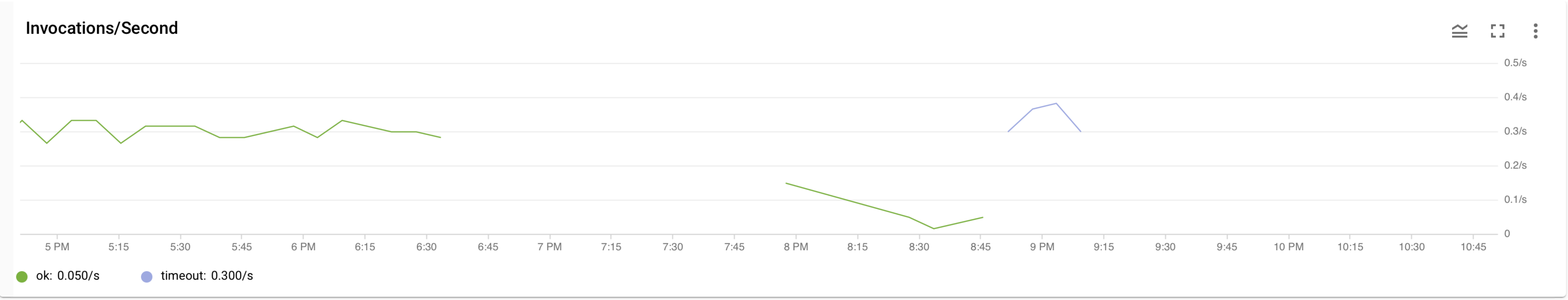


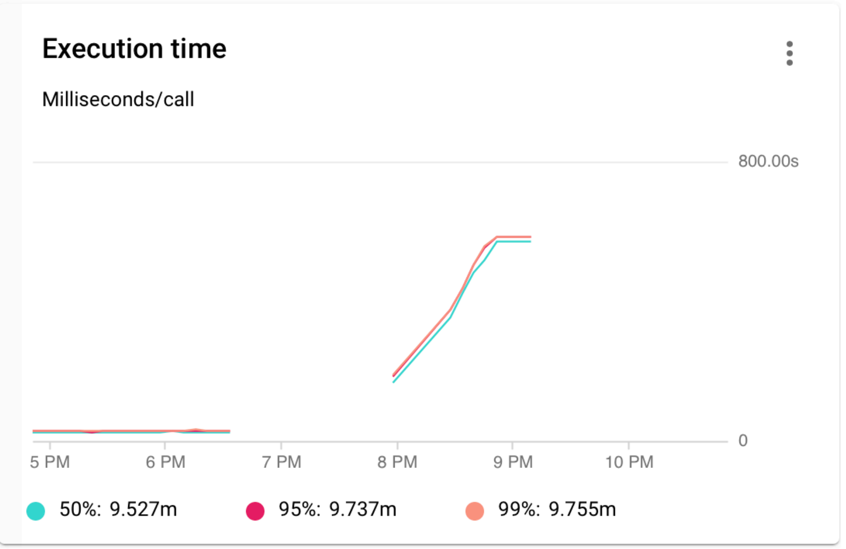


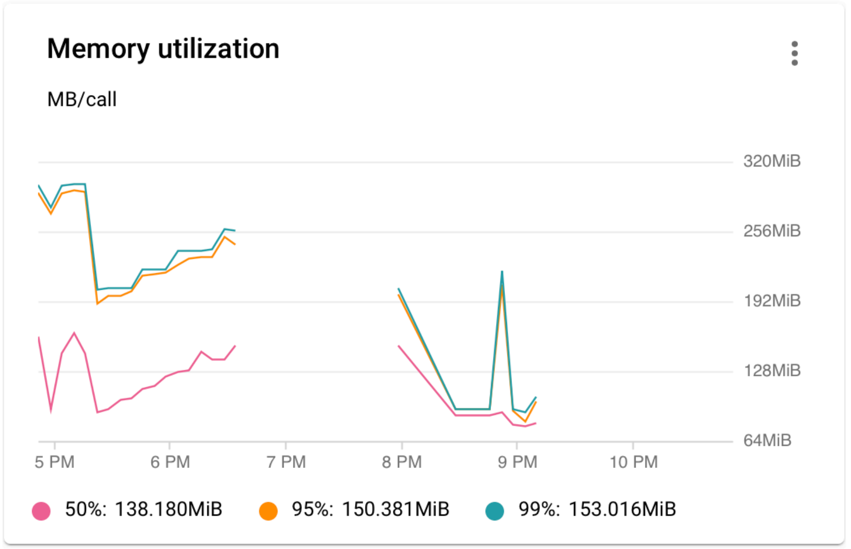


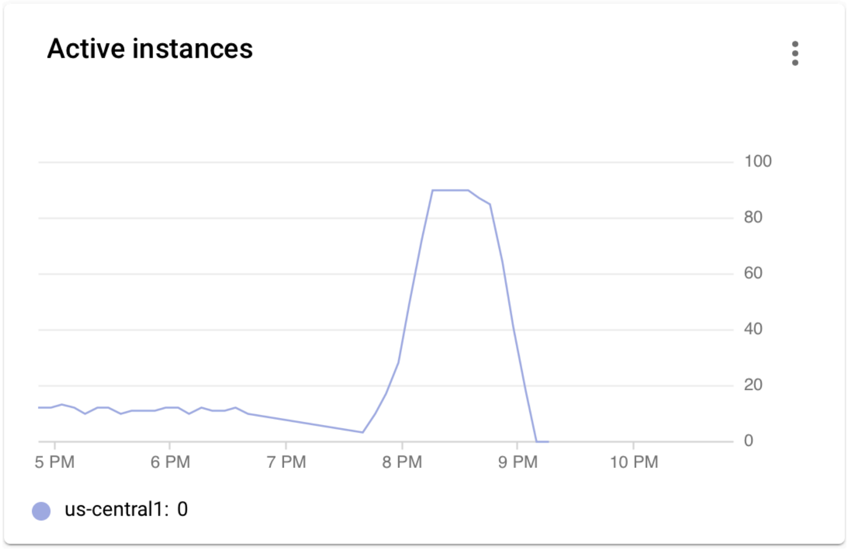
- 10x bulk tradeoff with only 4x total lead time: less cost as invocations same but only 1/3 instances
- Execution time cap limited by elasticsearch throughput – due to single VM instance (cluster better for higher scale for latency improvement) – cumulative write backpressure may cause cloud function timeout (9min) at large scale and large chunk size

Cloud Function stats – 100k vs. 1M records ingest (better strategy with higher bulk&batch)









- 10x bulk tradeoff with only 4x total lead time: less cost as invocations same but only 1/3 instances
- Execution time cap limited by elasticsearch throughput – due to single VM instance (cluster better for higher scale for latency improvement) – cumulative write backpressure may cause cloud function timeout (9min) at large scale and large chunk size

Setup 1/2

- Elasticsearch: Debian VM image, run following:
  - git clone https://github.com/chwongeric/cloud_function.git
  - Install by running: cloud_function/PubSub-Function-ES/setup/elastic_debian.sh
  - sudo vi /etc/elasticsearch/elasticsearch.yml (modify following 2 lines):
    ```
    network.host: 0.0.0.0
    cluster.initial_master_nodes: ["node-1"]
    ```
  - sudo systemctl start elasticsearch.service
  - curl http://<public_ip>:9200
- VPC network -> Firewall: add new ingress rule (elasticsearch) allow tcp:9200
- API: enable logging, build (for Cloud Function)
- Pubsub: create topic (name: inventory) and copy topic name - projects/<project-id>/topics/inventory
- Cloud Function: Trigger Topic: inventory, Mem allocated (128MB), Runtime: Python3.8, Ingress settings: allow all traffic
  - Runtime environment variable (convenient for passing constant like Elasticsearch endpoint etc)
    - ELASTICSEARCH_IP=34.71.9.193
  - Code: see under src dir
  - Test: see under test dir

Setup 2/2

- Cloud Shell: used for trigger sprayer and ES test:
  - $ gcloud config set project <project-id>
  - $ python3 -m pip install google.cloud
  - $ ./es_test.py -i <ip> -p 9200 -n True
  - $ ./generate.py –r 1000000
  - $ gunzip INVENTORY.json.gz
  - $ ./spray.py -c 100 –s 10000 (100 chunks of 10k rec msg)
  - $ ./es_test.py -i <ip> -p 9200 -s 20 (verify and query for result of 20)
  - $ ./es_test.py -i <ip> -p 9200 -n True (verify count)
  -