

Virtualization and Cloud Computing

CSL 7510

Live Price streaming of commodities with help of cloud technology system



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Instructor: Dr. Sumit Kalra

Team Members:

Sumit Keshari

M20CS021

Amaresh Chandra Singh

M20AIE212

Contributions: (By Sumit Keshari)

- Architecture modelling
- Implementation of the system on Google Cloud Computing Platform
- Building pipeline to fetch live metals data from API and dumping to the system implemented
- Writing code to fetch data from API provider and dump to the system database
- Analysis and comparison of the system performance
- Report making
- Presentation making

Contributions: (By Amaresh Chandra Singh)

- Buying google cloud computing resources

Live Price streaming of commodities with help of cloud technology system

Background:

"Cloud" is a concept connected to the concept of a distributed computer - a system in which software, hardware, and processes that make up an IT system can be deployed across locations around the world. With cloud-based computing, the network is generally regarded as the Internet, and the simple definition of computer cloud is the essential delivery of web capabilities, computer capabilities, storage, and online applications. Cloud based computing allows easy access to applications and data from anywhere in the world.

There was a client / server computer in front of the cloud computer that appeared in the central storage where all data and all controls reside on the server side. In this type of system, if one user wants to access certain data, the user needs to connect to the server to get proper access. Later, when multiple computers were connected together with shared resources, a distributed computer appeared in the image. In 1955, John MacCarthy who coined the term "Artificial Intelligence" (AI) suggested in a MIT speech that computing could be used and marketed as water or electricity based on everyday use.

The dream of a computer marketed as a reality came true when in 1999, salesforce.com began delivering apps to users using a simple website. In addition, in 2002, Amazon launched Amazon web services, providing services such as accounting, retention, and personal intelligence.

In 2009, Google apps also started to provide cloud computing enterprise applications, and in the same year Microsoft launched Windows Azure to provide various kinds of services. Some other major cloud service provider include:

- Apple, Citrix, IBM, Salesforce, Alibaba, Oracle cloud etc.

In this project, we implemented a system using google cloud computing platform which collects data using API called RapidAPI which gives live prices of metals like Gold, Silver, Platinum, Palladium in real time with minimum latency in 160+ currencies.

Problem Statement:

Cloud computing technology has removed Prior to the invention of cloud computing every hardware needed to be owned and maintained by each company which incurred huge costs on the operation. These systems require expensive hardware, software and costly infrastructure. In this project we implemented a system using Google Cloud computing platform which collects data using an API called RapidAPI which gives live prices of metals like Gold, Silver, Platinum, Palladium etc. in real time with minimum latency. We also guarantee this system can be easily modified to stream prices of different commodities items with less overhead. This project can be altered to be used in the different geographical locations without any topology alteration in the deployed solution.ed the barrier of publishing information from various sources to databases.

Methodology:

As the name of the project is **Live Price streaming of commodities with help of cloud technology system**, the methodology that we choose to implement our project is as follow:

1) Google cloud computing platform:

We implemented our project using google cloud computing platform. Google provides different types of subscription services based on different criteria. \$300 dollar worth credits are free to use and implement projects. We also used free \$300 dollar credits for our implementation.

Nevertheless, one can implement our approach on any cloud computing platform which hosts kubernetes and docker.

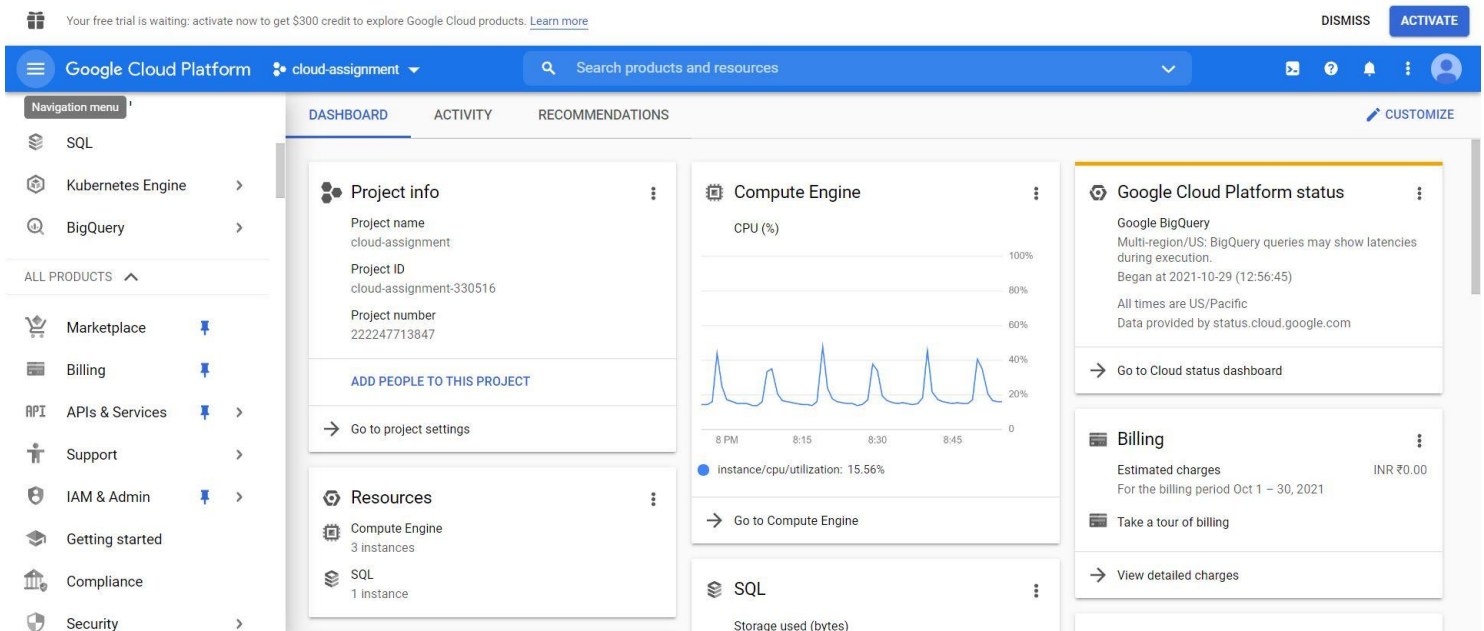


Figure: Google Cloud Computing Platform

2) Architecture:

The below figure defines the connection among the various services used to implement this project.

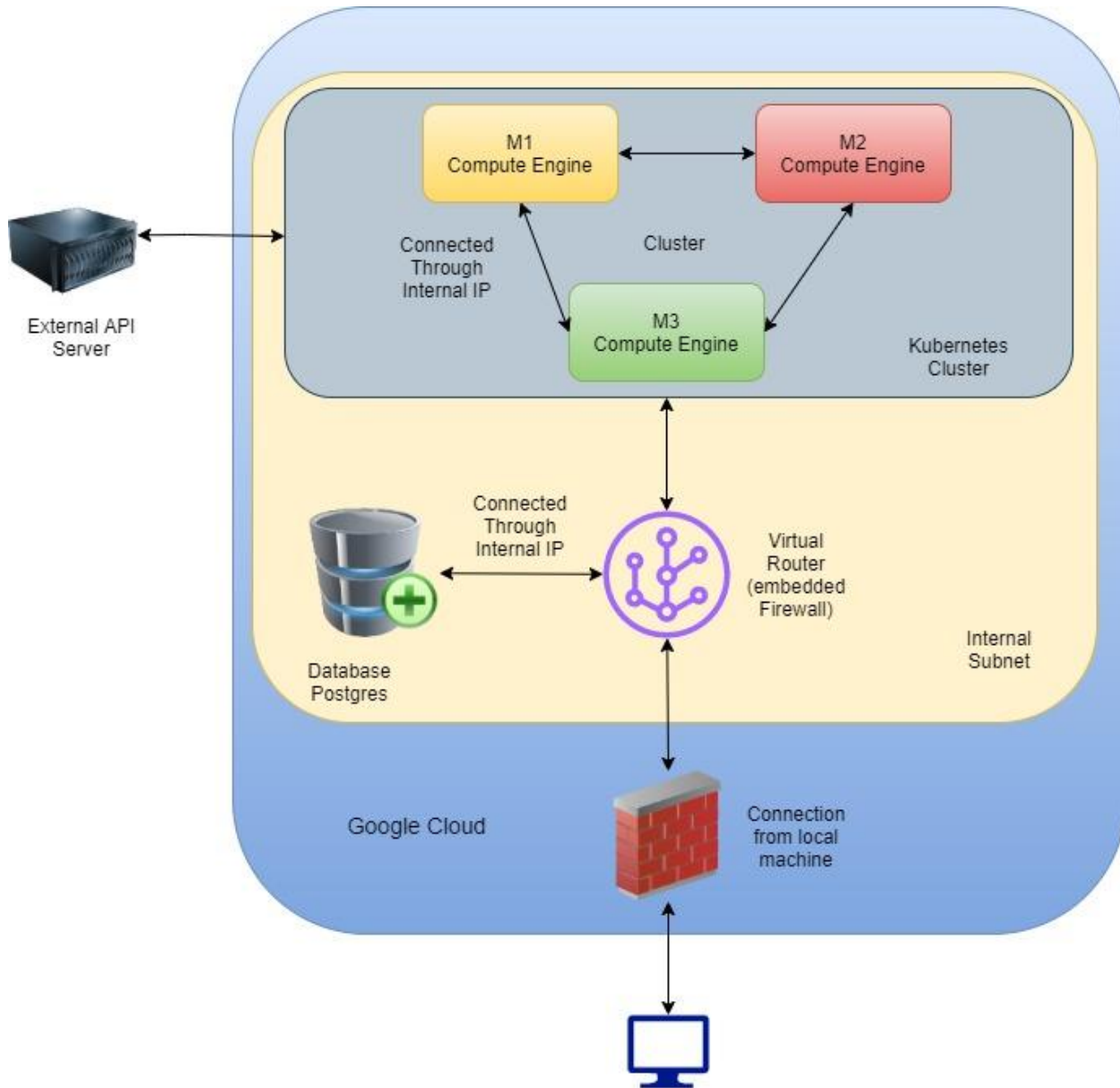


Fig: Architecture

We have used three instances of the compute engines from google cloud computing platform. These compute engines are connected in a subnet with

their internal IP addresses to facilitate orchestration of Docker images deployed and orchestrated by Kubernetes.

We have then connected this Subnet to a database. The database which we have used is POSTGRES, the reason being open source and free to use at any scale. We have connected this database to the cluster via virtual routing in which we have used public address allocation for the database to get it connected to the cluster. So in implementation compute engines connected internally via internal IP address and database connected to cluster via routing and mapping internal IP addresses of cluster versus public IP address of database.

To get connected this implementation of machines from our local machine to communicate, deploy, manage and monitor various services available in the cluster.

We are using free REST APIs to get the live data from the provider of commodity prices in real time. The APIs requests are limited to 50 requests/month. Hence, data collected is less in our implementation.

3) Automatic Trigger to push data:

The following python script we have used to automatically push the data to the data channel.

```
# -*- coding: utf-8 -*-
"""
@author: Sumit
"""
import time
import requests
import json
url = "https://live-metal-prices.p.rapidapi.com/v1/latest/XAU,XAU_OPEN,XAU_CHANGE,XAU_CHANGEPERCENT,XAU_1K,XAU_2K,XAU_3K,XAU_4K,XAU_5K,XAU_6K,XAU_7K,XAU_8K,XAU_9K,XAU_10K,XAU_11K,XAU_12K,XAU_13K,XAU_14K,XAU_15K,XAU_16K,XAU_17K,XAU_18K,XAU_19K,XAU_20K,XAU_21K,XAU_22K,XAU_23K,XAU_24K,XAG,XAG_OPEN,XAG_CHANGE,XAG_CHANGEPERCENT,PA,PL/INR/GRAM"
```

```

headers = {
    'x-rapidapi-host': "live-metal-prices.p.rapidapi.com",
    'x-rapidapi-key': "17db97a36dmsbdd3c3fe07e7936ap1f47c1jsnc3309bf1bace"
}
count = 1

#This loop will call api every 6 minutes. Thus, 10 api call in an hour.
#Infinite loop indicates automatic trigger of api call.
print("Startin posting data every ",5," seconds")
while(True):
    # API Call
    response = requests.request("GET", url, headers=headers)
    response = json.loads(response.text)
    timestamp = int(time.time())
    response["time"] = timestamp
    final_response = {"records": [{"key" : "recordKey" +
str(count),"value": response}]}
    final_response = json.dumps(final_response)
    final_response = final_response.replace("\\", "")
    headers = {
        'Content-Type': 'application/vnd.kafka.json.v2+json',
    }
    data = final_response
    #this statement is used to send data to the database.
    #We are port forwarding 8082 port from cluster to local machine.
    response =
requests.post('http://localhost:8082/topics/rapid.data.api',
headers=headers, data=data)
    count+=1
    time.sleep(5)

```


4) Cluster Details:

my-cluster

Your cluster has low resource requests; it may be under-utilized. [Autoscaling documentation](#) [VIEW DETAILS AND POSSIBLE ACTIONS](#)

DETAILS **NODES** STORAGE LOGS

Node Pools

Filter node pools

Name	Status	Version	Number of nodes	Machine type	Image type	Autoscaling
default-pool	Ok	1.22.2-gke.1901	3	n1-highmem-2	Container-Optimized OS with Containerd (cos_containerd)	Off

Nodes

Filter nodes

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
gke-my-cluster-default-pool-38848ba1-c1er	Ready	528 mCPU	1.93 CPU	487.59 MB	11.07 GB	0 B	0 B
gke-my-cluster-default-pool-38848ba1-y084	Ready	230 mCPU	1.93 CPU	283.12 MB	11.07 GB	0 B	0 B
gke-my-cluster-default-pool-38848ba1-zir8	Ready	510 mCPU	1.93 CPU	397.97 MB	11.07 GB	0 B	0 B

Figure: Compute Engine Instances

my-cluster

Your cluster has low resource requests; it may be under-utilized. [Autoscaling documentation](#) [VIEW DETAILS AND POSSIBLE ACTIONS](#)

DETAILS **NODES** STORAGE LOGS

Cluster basics

Name	my-cluster
Location type	Zonal
Control plane zone	us-central1-c
Default node zones	us-central1-c
Release channel	Rapid channel
Version	1.22.2-gke.1901
Total size	3
Endpoint	35.226.127.174

[Show cluster certificate](#)

Automation

Maintenance window	Any time
Maintenance exclusions	None
Upgrade notifications	Disabled
Vertical Pod Autoscaling	Enabled
Node auto-provisioning	Disabled
Autoscaling profile	Optimize utilization


Figure: Cluster Details

5) Script Used:

We have used various scripts to channelize the data to the database. All of the script along with proper documentation is attached along with the project. Please refer to the script for further details.

Results:

The results are shown below:



```
PS E:\API> python.exe .\try.py
Starting posting data every 5 seconds
API call number 1

API call number 2

API call number 3

API call number 4

API call number 5

API call number 6

Traceback (most recent call last):
  File ".\try.py", line 40, in <module>
    time.sleep(sleepTime)
KeyboardInterrupt
PS E:\API>
```

Figure: API Call

SQL select * from metal_prices_eur_euro changes;																															
SOURCE	VALID	BASEC	UNIT	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL																					
RS	DATE	TIME	UNIT	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL
PRICE	DATE	TIME	UNIT	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL	YEAR	MONTH	DATE	PRICE	CHANGE	RANGE	PL
1	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
2	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
3	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
4	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
5	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
6	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
7	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
8	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
9	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
10	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
11	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
12	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
13	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
14	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
15	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
16	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
17	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
18	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
19	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
20	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
21	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
22	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
23	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
24	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
25	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
26	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
27	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
28	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
29	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
30	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
31	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
32	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
33	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
34	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
35	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
36	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
37	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1
38	1999	1	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01	4336.1	5.436	1180-7361	1	1999	1	1999-01-01											

Analysis of the System performance:

CPU Usage:

The chart displays CPU usage percentage over a period of 6 hours. The usage starts at approximately 61% at 12:00, fluctuates between 58% and 62% until 13:00, then rises to a peak of about 62.5% at 13:45. It then declines to a low of about 57% at 16:00, before rising again to approximately 62.5% by 17:45. A green dot marks the data point at 13:00.

Time	CPU Usage (in %)
12:00	61.0
12:15	59.5
12:30	60.5
12:45	60.0
13:00	60.5
13:15	61.5
13:30	61.0
13:45	62.5
14:00	61.5
14:15	60.5
14:30	59.5
14:45	60.5
15:00	59.5
15:15	59.0
15:30	57.5
15:45	58.5
16:00	57.0
16:15	58.0
16:30	57.5
16:45	57.0
17:00	57.5
17:15	58.5
17:30	61.0
17:45	62.5

Figure: CPU usage

Throughput:

Below is the graph for API throughput bytes/second. The throughput performance at starting is quite good. On the right hand side of the figure, we can see that there is low throughput. This might be due to the fact that at that time the API provider is having a high number of API calls. There could be other reasons for this like network fluctuations etc. As we are completely relying on API provider for the data, we think there might be some heavy load issues on the providers side.



Figure: Throughput

Network Latency:

The following graph shows the network latency among compute engines. The dip on the graph is due to a huge number of network calls between our compute engine instances.

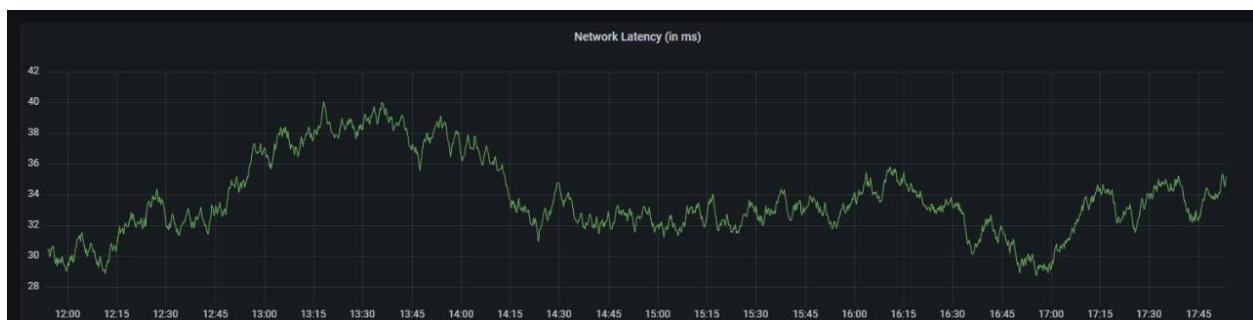


Figure: Network Latency

Database usage:

There is low database usage as we move towards the right side of the graph. There can be multiple reasons for the same. Some few major reasons for the same are: due to low cpu utilization of our compute engine instances, low throughput of the API calls, network latency, large number of network calls between the compute engine instances etc.

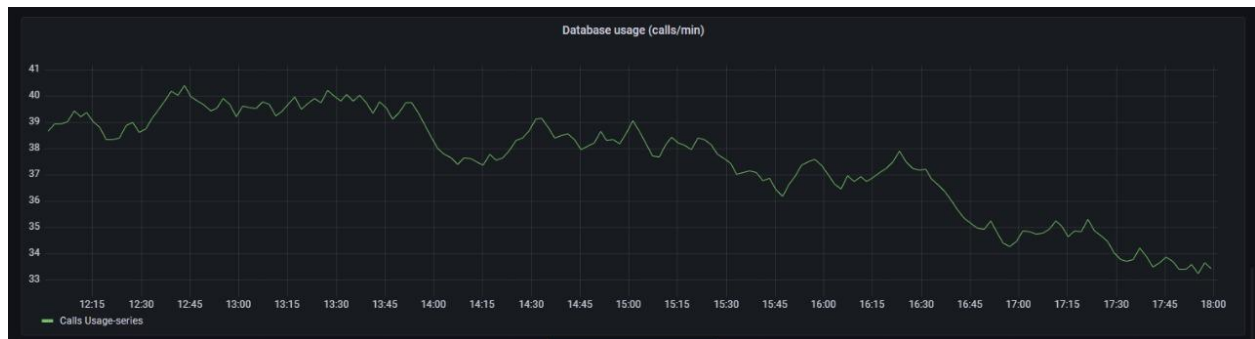


Figure: Database Usage

Comparison:

We tried to analyse our system with different industry standard streaming tools like **Google Cloud Dataflow, Azure Stream Analytics, Apache Stream Analytics, Apache Storm** etc. Since these are paid softwares and require credit to access, we have not been able to install these software and compare our own architecture.

But one thing we are sure of is that our system performance is much lower in comparison to these standard systems. Our system has low cpu utilization, low database utilization, high network latency etc. in comparison to the standard system. According to the documentation of Google Cloud Dataflow it has latency of **5-10 milliseconds** on average for the same data whereas our system has a latency of **110 milliseconds** on average for the same data. We have tried to implement our own system and we are looking forward to improving it in future.

Future Directions:

In this project we have implemented live streaming of commodities with the help of a cloud technology system. Future works can include implementation of automated recovery, managing of multiple api calls from multiple resources, autoscaling of systems and visualization of the data stored in the database with proper insights.