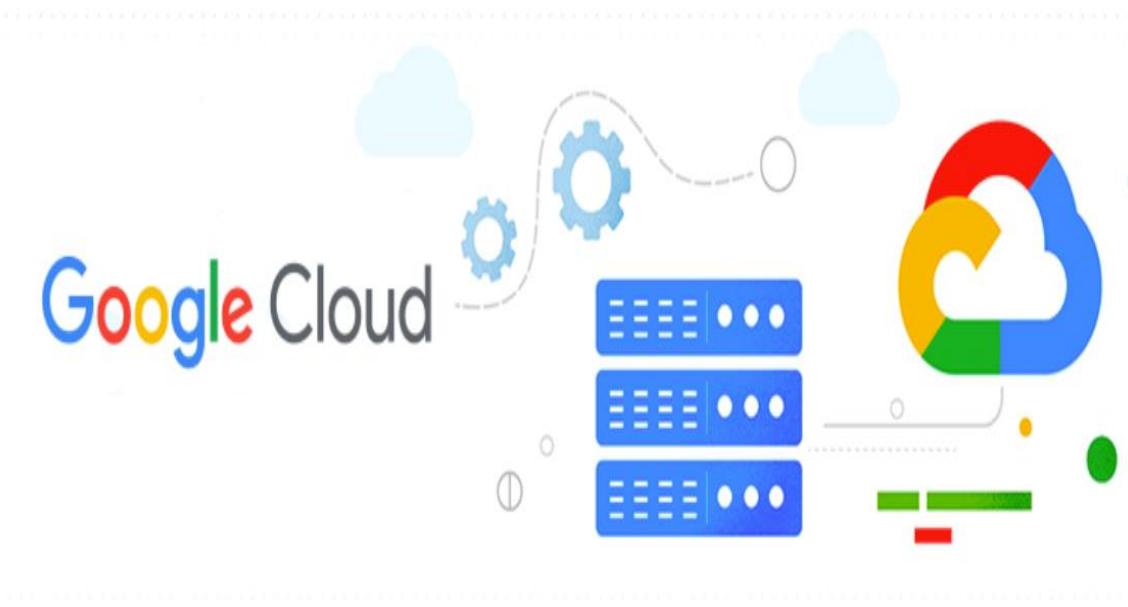


# **CREATE A CLOUD FUNCTION TO PULL A MESSAGE FROM PUB/SUB**

A Project Report Submitted in the fulfilment of the requirements for  
**Final-Project Evaluation**



**Name:** Sanjeev M

**Employee id:** 2320436

**GenC Intern**



## **CONTENTS**

- 1. OBJECTIVE**
- 2. INTRODUCTION**
- 3. REAL TIME USE CASE**
- 4. PLATFORM USED**
- 5. PROJECT REQUIREMENT**
- 6. IMPLEMENTATION**
- 7. CONCLUSION**

## **1.OBJECTIVES**

- The objective of this project is to build a real-time stock price monitoring system that alerts users when a stock price crosses a certain threshold.
- The system will use Google Cloud Pub/Sub to stream real-time stock price updates and Google Cloud Functions to analyse and send alerts.
- By accomplishing these objectives, this project aims to empower organizations to leverage GCP's storage capabilities effectively, driving operational efficiency, and enhancing data governance practices.

## **2.INTRODUCTION**

In the ever-expanding landscape of cloud computing, efficient data storage management is paramount for organizations to balance performance, accessibility, and cost-effectiveness. Google Cloud Platform (GCP) offers a comprehensive suite of storage solutions tailored to meet diverse needs, including Standard and Archive buckets, each optimized for specific use cases.

Google Cloud Platform (GCP) offered by Google, is a suite of cloud computing services that Provides a series of modular cloud services including computing, data storage, data analytics, and machine learning, artificial intelligence alongside a set of management tools.

GCP run on the same infrastructure that Google uses internally for its own products, such as Google Search and YouTube.

GCP is designed to be scalable, reliable, and secure. It is also cost-effective, with a pay-as-you-go pricing model.



Here are some key points about GCP:

1. **GCP Services:** GCP offers a wide range of infrastructure and application services that can be accessed on-demand. It enables users to build, deploy, and scale applications seamlessly while taking advantage of Google's powerful and reliable infrastructure.
2. **Global Resources:** GCP consists of physical assets (such as computers and hard disk drives) and virtual resources (such as virtual machines) distributed across data centres worldwide. These resources are organized into regions and zones, providing redundancy and reduced latency for better performance.

**Services and Integration:** When you develop applications on GCP, you combine various services to create the infrastructure you need. GCP services include compute, storage, databases, machine learning, and more. Additionally, GCP integrates seamlessly with other Google Cloud products.

## **Benefits of using GCP:**

There are many benefits to using GCP, including:

- **Scalability:** GCP can scale up or down to meet the demands of your application.

- **Reliability:** GCP is designed to be highly reliable, with a 99.9% uptime guarantee.
- **Security:** GCP is one of the most secure cloud platforms available, with a variety of security features to protect your data.
- **Cost-effectiveness:** GCP is a cost-effective cloud platform, with a pay-as-you-go pricing model.

Getting started with GCP is easy. You can create a free account and start using GCP services immediately.

To create a GCP account, visit the GCP website and click on the "Create an account" button. You will need to provide your name, email address, and a password.

Once you have created an account, you can start using GCP services. To learn more about GCP, you can visit the GCP documentation website.

Common use cases for GCP

GCP can be used for a wide variety of applications, including:

- Web and mobile applications: GCP can be used to host web and mobile applications.
- Data storage and analytics: GCP can be used to store and analyse data.
- Machine learning and artificial intelligence: GCP can be used to develop and deploy machine learning and artificial intelligence models.

### **3.REAL TIME USE CASE:**

### **REAL-TIME STOCK PRICE MONITORING**

Build a real-time stock price monitoring system that alerts users when a stock price crosses a certain threshold. Use Cloud Pub/Sub to stream real-time stock price updates and Cloud Functions to analyse and send alerts.

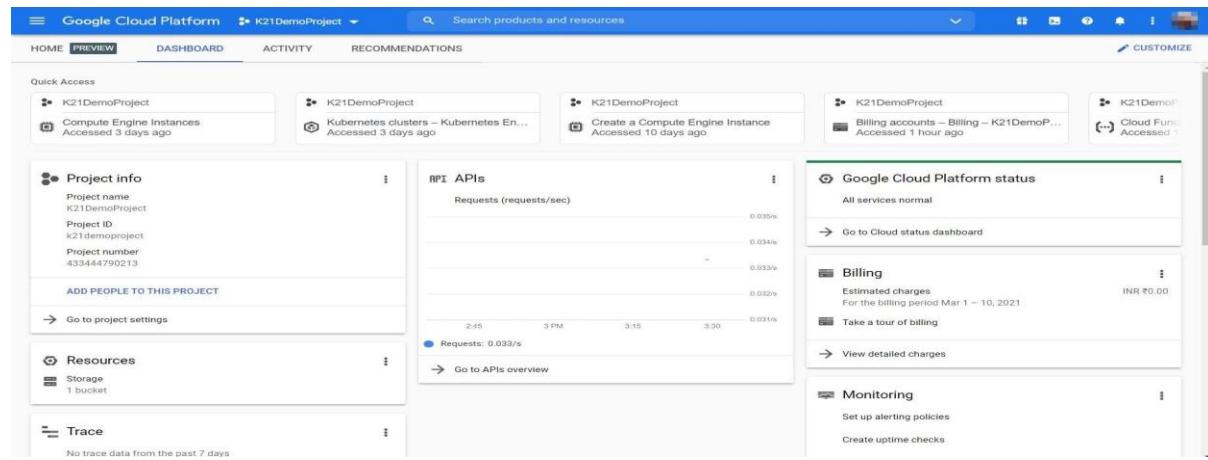
#### **Steps:**

- Fetching the stock price of Infosys by parsing the price html class and pushing it into up Cloud Pub/Sub.
- Used cloud function to analyse whether the stock price have raised above the threshold value.
- Send alerts via email by Cloud Function.

### **4. PLATFORM USED:**

### **GOOGLE CLOUD PLATFORM INTERFACE:**

#### **Web Console:**



**Google Cloud Platform (GCP) Account:** Sign up for a GCP account if you don't have one already. This provides access to GCP's storage services, including Cloud Storage for creating buckets and managing data.

**Google Colab:**



Google Colab (short for Collaboratory) is a cloud based Jupyter notebook environment that allows you to write and execute Python code. It's particularly useful for data science, machine learning, and collaborative projects.

**Google Looker:**



- Google Looker is a business intelligence and data analytics platform developed by Looker Data Sciences, Inc., which was acquired by Google Cloud in 2020.

- Looker provides a comprehensive suite of tools for data exploration, visualization, and reporting, enabling organizations to derive insights from their data.
- It offers a semantic modelling layer that allows users to define metrics and dimensions in a business-friendly language, facilitating collaboration between technical and non-technical users.
- Google Looker is often used by businesses to analyse data from various sources, create interactive dashboards, and make data-driven decisions.
- With the acquisition by Google Cloud, Looker is integrated with Google's cloud infrastructure and services, providing additional capabilities for data processing and analysis.

## **5.PROJECT REQUIREMENTS**

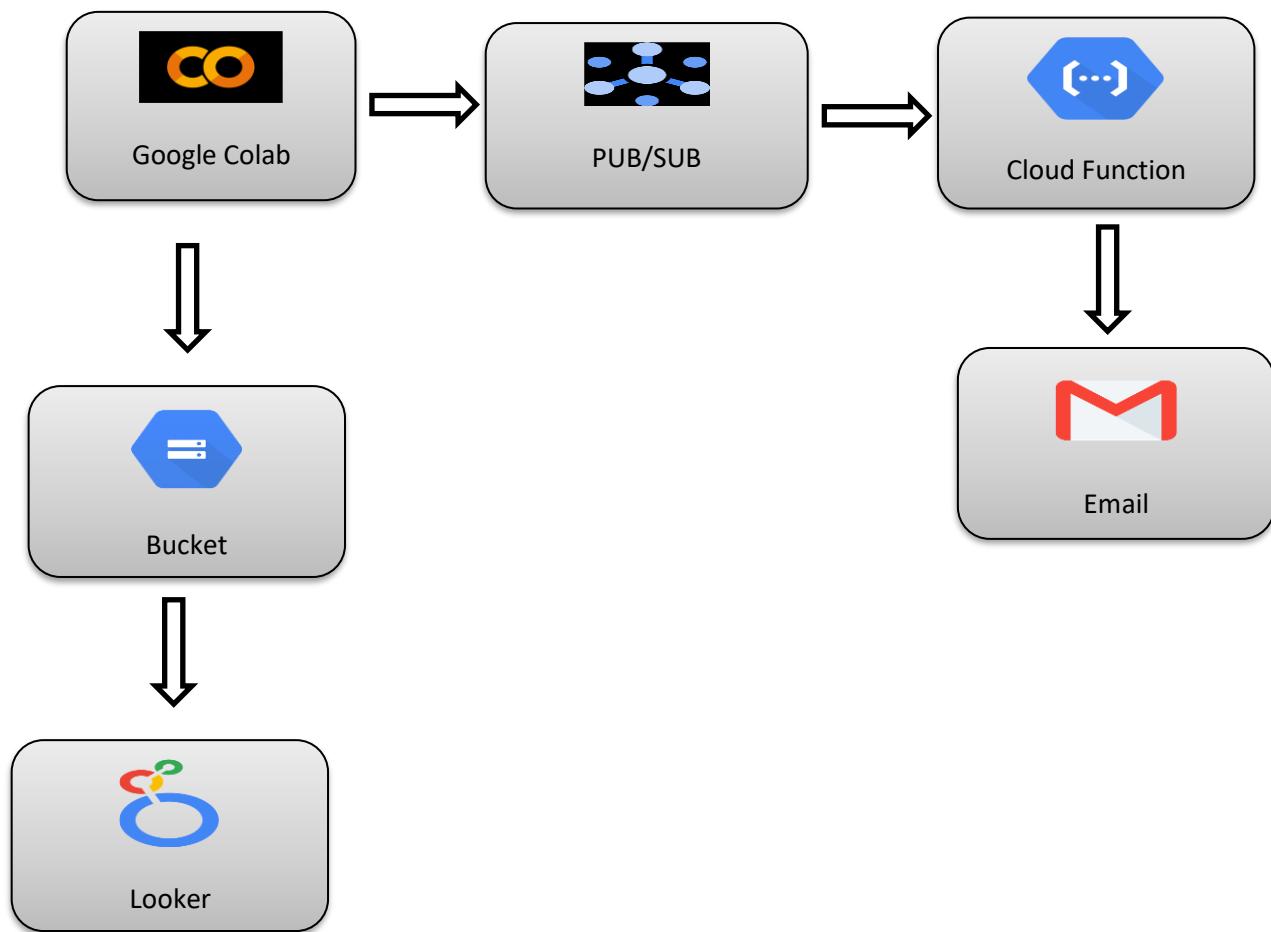
### **Cloud Pub/Sub:**

- Pub/Sub, short for Publisher/Subscriber is a messaging service provided by Google cloud platform (GCP) that enables asynchronous communication between independent application.
- Publishers send messages to a topic, and Subscribers receive those messages from the topic they are interested in.
- This decoupling allows for scalable and flexible communication between components of an application or between different applications altogether.
- Pub/Sub supports durable messaging, meaning messages are retained even if subscribers are temporarily offline, ensuring reliable delivery.
- Additionally, it integrates seamlessly with other GCP services, making it an essential tool for building distributed and event-driven architectures in the cloud.

## **CLOUD FUNCTION:**

- Cloud Functions is a serverless computing service provided by Google Cloud Platform (GCP) that allows developers to deploy and run event-driven functions in a fully managed environment.
- With Cloud Functions, developers can write lightweight code snippets in languages such as Node.js, Python, Go, and more, and deploy them without worrying about managing infrastructure.
- These functions are triggered by events from various GCP services, such as Cloud Storage, Cloud Pub/Sub, HTTP requests, and many others.
- Cloud Functions automatically scales up or down to match the incoming workload, ensuring optimal performance and cost-effectiveness.
- It is also called as a Function as a Service (FaaS).

## 6. IMPLEMENTATION



Firstly, we are creating a topic named as “final-demo” with default subscription.

The screenshot shows the 'Create topic' interface in the Google Cloud Pub/Sub console. The 'Topic ID' field contains 'final-demo'. The 'Add a default subscription' checkbox is checked. In the 'Encryption' section, the 'Google-managed encryption key' option is selected. A 'CREATE' button is at the bottom.

Here, default subscription is named as “final-demo-sub”.

The screenshot shows the 'final-demo' topic detail page. It lists one subscription: 'final-demo-sub' with 'Subscription name' 'projects/cts-0023/subscriptions/final-demo-sub' and 'Project' 'cts-0023'. A success message box at the bottom states 'A new topic and a new subscription have been successfully created.'

The topic named “final-demo” is created successfully.

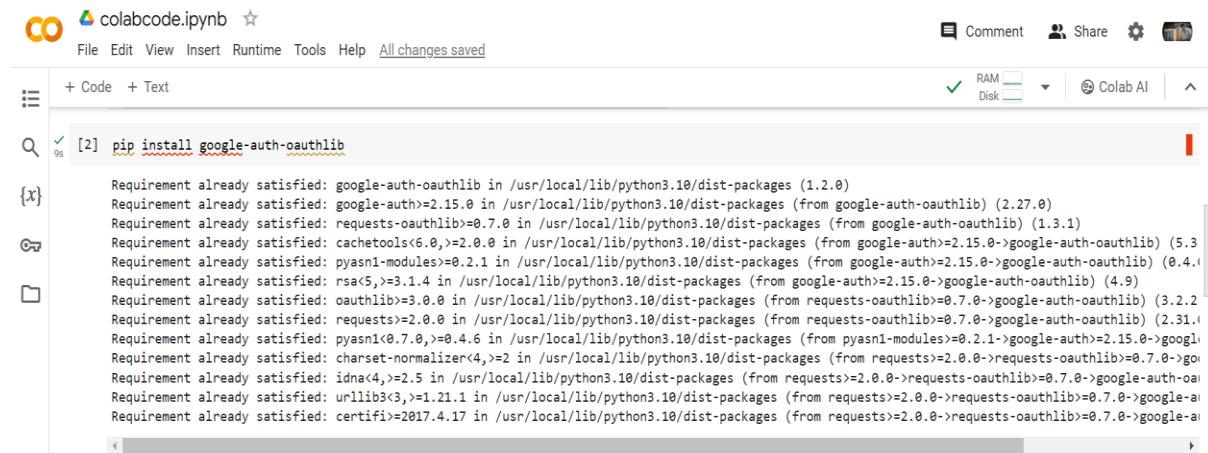
The screenshot shows the Google Cloud Pub/Sub console interface. In the top navigation bar, the project is set to "cts-0023" and the service is "pubsub". The main area displays a table titled "Topics" under the "LIST" tab. A new row is added for the topic "final-demo", which is "Google-managed" and belongs to the project "projects/cts-0023/topics/final-demo". A success message at the bottom states: "A new topic and a new subscription have been successfully created." The status bar at the bottom right shows the date as 22-04-2024 and the time as 12:06.

This line installs the `google.cloud.pubsub` package, which is required for interacting with Google Cloud Pub/Sub.

A Jupyter Notebook cell is shown with the command `pip install google.cloud.pubsub`. The output of the command shows the package being downloaded and installed along with its dependencies. The notebook interface includes code and text tabs, and a sidebar with various icons.

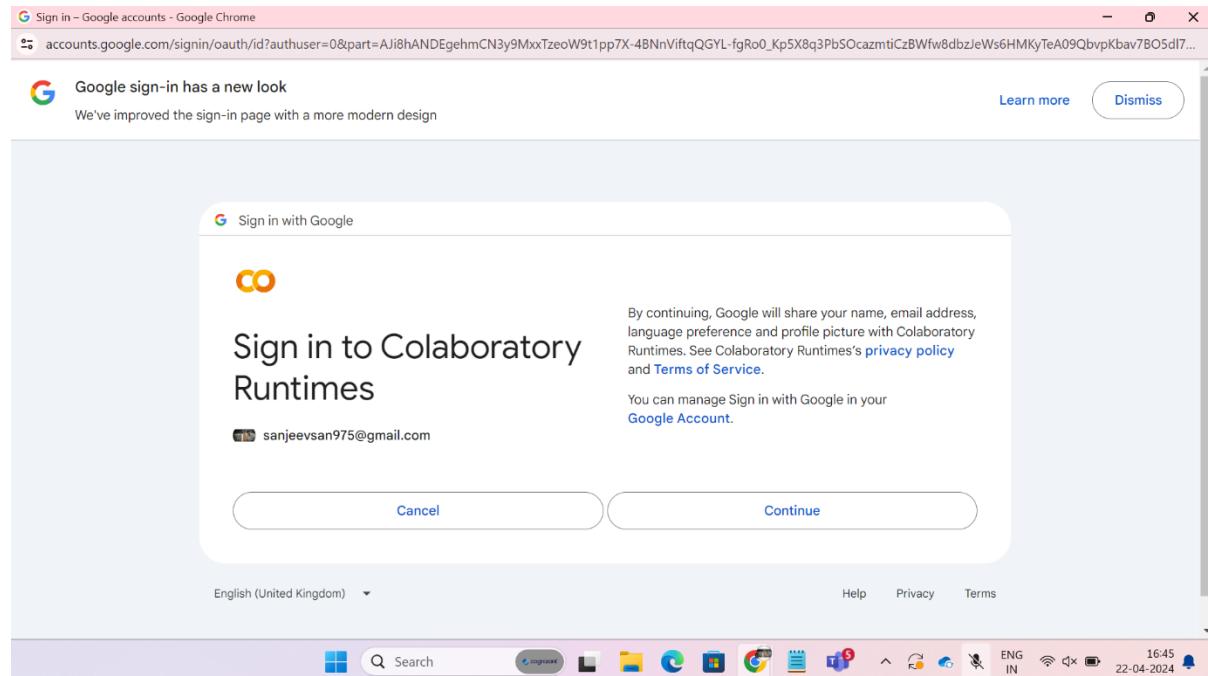
```
[1]: pip install google.cloud.pubsub
Collecting google.cloud.pubsub
  Downloading google_cloud_pubsub-2.21.1-py2.py3-none-any.whl (273 kB)
    273.2/273.2 kB 2.0 MB/s eta 0:00:00
Requirement already satisfied: grpcio<2.0.dev,>=1.51.3 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (1.62.1)
Requirement already satisfied: google-auth<3.0.dev,>=2.14.1 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (2.27.0)
Requirement already satisfied: google-api-core[grpc]=2.0.*,!2.1.*,!2.10.*,!2.2.*,!2.3.*,!2.4.*,!2.5.*,!2.6.*,!2.7.*,!2.8.*,!2.9.*,<3.0.dev
Requirement already satisfied: proto-plus<2.0.dev,>=1.22.0 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (1.23.0)
Requirement already satisfied: protobuf!=3.,!20.0,!3.,!20.1,!4.,!21.0,!4.,!21.1,!4.,!21.2,!4.,!21.3,!4.,!21.5,<5.0.dev,>=3.19.5 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (3.19.5)
Requirement already satisfied: grpc-google-iam-v1<1.0.dev,>=0.12.4 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (0.13.0)
Requirement already satisfied: grpcio-status!=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (1.48.2)
Requirement already satisfied: googleapis-common-protos<2.0.dev,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]==2.0.*)
Requirement already satisfied: requests<3.0.0.dev,>=2.18.0 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]==2.0.*)
Requirement already satisfied: cachetools<6.0,>2.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.dev,>=2.14.1>google.cloud.pubsub)
Requirement already satisfied: pyasn1-modules==0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.dev,>=2.14.1>google.cloud.pubsub)
Requirement already satisfied: rsa<5,>3.1.4 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules==0.2.1>google-auth<3.0.dev,>=2.14.1>google.cloud.pubsub) (4.9)
Requirement already satisfied: charset-normalizer<4,>2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev,>=2.18.0>google-api-core)
Requirement already satisfied: idna<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules==0.2.1>google-auth<3.0.dev,>=2.14.1>google.cloud.pubsub) (4.9)
Requirement already satisfied: urllib3<3,>1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev,>=2.18.0>google-api-core[grpc])
Requirement already satisfied: certifi!=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev,>=2.18.0>google-api-core[grpc])
Installing collected packages: google.cloud.pubsub
Successfully installed google.cloud.pubsub-2.21.1
```

This line installs the `google-auth-oauthlib` package, which is required for handling authentication with Google services.



```
[2] pip install google-auth-oauthlib
Requirement already satisfied: google-auth-oauthlib in /usr/local/lib/python3.10/dist-packages (1.2.0)
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib) (2.27.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib) (1.3.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib) (5.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-auth-oauthlib) (0.4.1)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-auth-oauthlib) (4.9)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.2)
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib) (2.31.1)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth>=2.15.0->google-auth-oauthlib) (0.4.7)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.1)
```

Here, Sign in page of colab will pop up as a part of Authentication from the OAuth package.



- Here, it initiates the authentication process for the user, which is necessary for accessing Google Cloud services securely.
- Next line imports the `requests` module, which is used for making HTTP requests to fetch web pages or interact with web APIs.
- Then next line imports the `BeautifulSoup` class from the `bs4` (Beautiful Soup 4) package, which is used for parsing HTML and XML documents.
- Next line imports the `time` module, which is used for adding time delays or getting current time information.
- Then next line imports the `pubsub\_v1` module from the `google.cloud` package, which provides functionality for interacting with Google Cloud Pub/Sub.
- Then next line defines a function named `scrape\_price` that takes two parameters: `url` and `class\_name`. This function is used to scrape price data from a webpage.
- Next line extracts the price data from the HTML content, converts it to a floating-point number, and removes any extraneous characters or formatting.

```
[3]: from google.colab import auth
      from google.auth.transport.requests import Request
      from google.oauth2 import id_token
```

```
[5]: auth.authenticate_user()
```



The screenshot shows a Google Colab notebook titled "1.ipynb". The code cell contains the following Python script:

```
import requests
from bs4 import BeautifulSoup
import time
from google.cloud import pubsub_v1

# Function to scrape price data
def scrape_price(url, class_name):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Check if the element with the specified class name exists
    if soup.find(class_=class_name) is not None:
        price = float(soup.find(class_=class_name).text.strip()[1:].replace(",",""))
        return price
    else:
        return None

# Original code variables
ticker = "INFY"
url = f"https://www.google.com/finance/quote/{ticker}:NSE?hl=en"
class_name = "YMLKec fxKbKc"

# Demo code variables
```

The status bar at the bottom indicates "Connected to Python 3 Google Compute Engine backend". The taskbar at the bottom of the screen shows various application icons.

- Then it assigns the value `'"INFY"'` to the variable `ticker`, representing the stock ticker symbol.
- Then it constructs the URL for fetching the stock price data from Google Finance, using the ticker symbol `'"INFY"'`.
- Then it assigns the class name of the HTML element containing the stock price data to the variable `class\_name`.
- Then next it creates a publisher client object for interacting with Cloud Pub/Sub.
- Then it calls the `scrape\_price` function to fetch the stock price data from the specified URL using the given class name.

The screenshot shows a code editor window with a light gray background. On the left, there is a vertical scroll bar. The main area contains the following Python code:

```
    return price
else:
    return None
# Original code variables
ticker = "INFY"
url = f'https://www.google.com/finance/quote/{ticker}:NSE?hl=en'
class_name ="YMIKec fxkbKc"

# Demo code variables
project_id = "cts-0023"
topic_name = "final_project"

# Create a PublisherClient
publisher = pubsub_v1.PublisherClient()

# Create the topic path
topic_path = publisher.topic_path(project_id, topic_name)

# Main loop
for i in range(3):
    # Scrape price
    price = scrape_price(url, class_name)
    if price is not None:
        # Prepare message
```

At the bottom of the code editor, there is a status bar with the text "✓ 5m 2s completed at 5:40PM". To the right of the status bar are several small icons: a green checkmark, a play button, a refresh arrow, a magnifying glass, a gear, a document, a trash can, and a vertical ellipsis.

- The line encodes the message data as bytes before publishing it to Cloud Pub/Sub.
- Then next it publishes the message data to the Cloud Pub/Sub topic.
- Then next it waits for the message to be published before proceeding.
- Then it pauses the execution of the program for 100 seconds before the next iteration of the loop. This is used to limit the frequency of price updates to once every 100 seconds.

```

+ Code + Text
  ⚡ # Main loop
  for i in range(3):
    # Scrape price
    price = scrape_price(url, class_name)
    if price is not None:
      # Prepare message
      message = price

      # Publish message to Pub/Sub
      message_data_bytes = str(message).encode("utf-8")
      future = publisher.publish(topic_path, data=message_data_bytes)
      future.result() # Wait for the message to be published
      print(price)
    else:
      print(f"Price for {ticker} not found")

    # Wait for 600 seconds
    time.sleep(100)
  ...
  1421.45
  1420.5

```

After fetching the stock price, we are selecting the subscription ID to where the message must be sent.

The screenshot shows the Google Cloud Platform interface for the Pub/Sub service. A topic named "final-demo" is selected. A modal dialog is open, prompting the user to "CREATE A SUBSCRIPTION". The subscription path is set to "projects/cts-0023/subscriptions/final-demo-sub". The "MESSAGES" tab is active in the topic view. The left sidebar shows other options like Subscriptions, Snapshots, and Schemas.

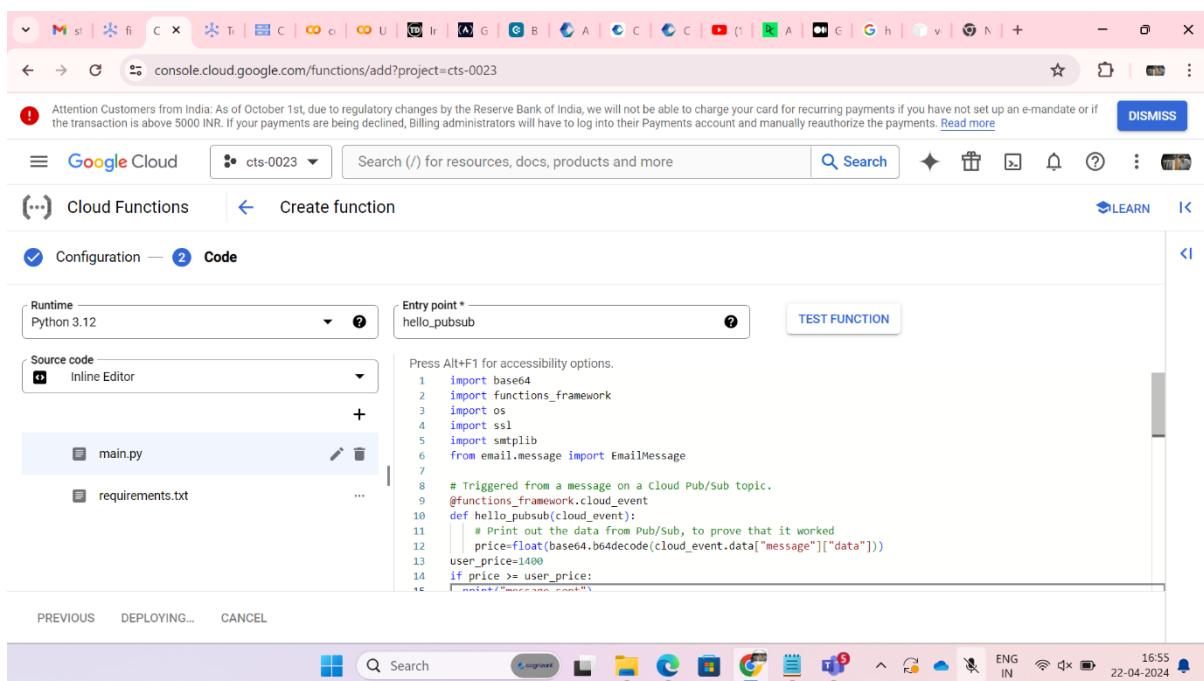
The INFY stock price is published in topic messages.

The screenshot shows the Google Cloud Pub/Sub interface. On the left, a sidebar menu is open under 'Pub/Sub' with 'Topics' selected. The main area displays the 'final-demo' topic, which was created in project 'cts-0023'. The 'MESSAGES' tab is active, showing two messages with their publish times, attribute keys, and message bodies. Both messages have an 'ACK' status. The interface includes standard navigation and search bars at the top and bottom.

Then, we built a cloud function named “final-function” with trigger type “cloud pub/sub” with the respective topic.

The screenshot shows the Google Cloud Functions interface. A new function is being created with the name 'final-function'. The 'Basics' section is filled with environment '2nd gen', function name 'final-function', and region 'asia-south1 (Mumbai)'. In the 'Trigger' section, the 'Trigger type' is set to 'Cloud Pub/Sub' and the 'Cloud Pub/Sub topic' is set to 'projects/cts-0023/topics/final-demo'. The 'NEXT' button is visible at the bottom left.

- Here, we defined a cloud function .
- Initially, Base64 is a binary-to-text encoding scheme used to encode data.
- Functions\_framework is a framework for writing lightweight functions as part of Google Cloud Functions
- Then we imported the EmailMessage class from the email.message module, which is used to represent an email message.
- Then it defines a function named `hello\_pubsub` which takes a `cloud\_event` as its parameter. This function will be triggered when a message is published to a Cloud Pub/Sub topic.



The screenshot shows the Google Cloud Functions 'Create function' interface. At the top, there's a navigation bar with various icons and a search bar. Below it, a message about payment changes from India is displayed. The main area has tabs for 'Cloud Functions' and 'Create function'. The 'Code' tab is selected, indicated by a blue checkmark. On the left, the 'Runtime' is set to 'Python 3.12'. The 'Source code' section contains an 'Inline Editor' for 'main.py' and a 'requirements.txt' file. The 'main.py' code is as follows:

```

Runtime Python 3.12
Entry point * hello_pubsub
TEST FUNCTION

Press Alt+F1 for accessibility options.

1 import base64
2 import functions_framework
3 import os
4 import ssl
5 import smtplib
6 from email.message import EmailMessage
7
8 # Triggered from a message on a Cloud Pub/Sub topic.
9 @functions_framework.cloud_event
10 def hello_pubsub(cloud_event):
11     # Print out the data from Pub/Sub, to prove that it worked
12     price=float(base64.b64decode(cloud_event.data["message"]["data"]))
13     user_price=1400
14     if price >= user_price:
15         print("Message received")

```

At the bottom, there are buttons for 'PREVIOUS', 'DEPLOYING...', and 'CANCEL', along with a system tray at the very bottom.

- This line decodes the data from the Cloud Pub/Sub message, which is encoded in base64 format. It then converts it to a float, assuming the data represents a numeric value like a stock price.
- Next it checks if the price extracted from the Pub/Sub message is greater than or equal to a threshold price (`user\_price`). If the condition is true, it proceeds to send an alert.

The screenshot shows the Google Cloud Functions code editor for a project named 'cts-0023'. The 'Code' tab is selected. The runtime is set to Python 3.12. The entry point is 'hello\_pubsub'. The source code is in an inline editor, showing a script named 'main.py' and a requirements file. The code itself is as follows:

```

Runtime: Python 3.12
Entry point: hello_pubsub
Source code: Inline Editor
main.py
requirements.txt

Press Alt+F1 for accessibility options.
def hello_pubsub(cloud_event):
    # Print out the data from Pub/Sub, to prove that it worked
    price=float(base64.b64decode(cloud_event.data["message"])["data"])
    user_price=1400
    if price > user_price:
        print("message sent")
        email_sender = 'devagcppractice@gmail.com'
        email_password = 'solv evhg amvy wuq' # Fetching the password from environment variable
        email_receiver = 'sanjeevan97@gmail.com'
        subject = 'Alert:stock Trigger'
        body =f"""\nDear Investor,\n\nWe are writing to inform you that the stock value of Infosys has exceeded the threshold you set in your notification.\nGiven this development, we recommend reviewing your investment strategy and considering any necessary actions in this regard.\nPlease let us know if you require further assistance or have any questions regarding this alert."""

```

At the bottom, there are buttons for 'PREVIOUS', 'DEPLOYING...', and 'CANCEL'.

The screenshot shows the Google Cloud Functions code editor with the code modified. The 'Code' tab is selected. The runtime is set to Python 3.12. The entry point is 'hello\_pubsub'. The source code is in an inline editor, showing the same main.py file. The code now includes the final email sending logic:

```

Runtime: Python 3.12
Entry point: hello_pubsub
Source code: Inline Editor
main.py
requirements.txt

Press Alt+F1 for accessibility options.
Please let us know if you require further assistance or have any questions regarding this alert."""
23
24
25
26
27
28
29
30
31
32
33
34
35

email=EmailMessage()
em['From'] = email_sender
em['To'] = email_receiver
em['Subject'] = subject
em.set_content(body)

context = ssl.create_default_context()

with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
    smtp.login(email_sender, email_password)
    smtp.send_message(em)

```

At the bottom, there are buttons for 'PREVIOUS', 'DEPLOYING...', and 'CANCEL'.

The cloud function got deployed.

The screenshot shows the Google Cloud Functions console. At the top, there is a message about regulatory changes in India regarding recurring payments. Below it, the 'Cloud Functions' section is visible, with 'Function details' selected. A green icon indicates the function is active. The function name is 'function-final' (2nd gen), deployed at 22 Apr 2024, 17:08:44. The URL is https://asia-south1-cts-0023.cloudfunctions.net/function-final. A 'Powered by Cloud Run' badge is present. The interface includes tabs for METRICS, DETAILS, SOURCE, VARIABLES, TRIGGER, PERMISSIONS, LOGS, and TESTING. Under METRICS, there are two charts: 'Invocations/Second' and 'Execution time'. The 'Invocations/Second' chart shows a single data point at 10ms. The 'Execution time' chart shows a single data point at 10ms. The bottom of the screen shows the browser's address bar with the full URL and the system tray with the date and time (22-04-2024, 17:08).

When the stock price exceeded the threshold value then alert mail got triggered with a message.

The screenshot shows the Gmail inbox. The 'Compose' button is visible. The 'Inbox' tab is selected, showing 2,172 messages. One message is highlighted: 'Alert:Stock Trigger' from 'devagcppraccise@gmail.com' (sent 17:35, 6 minutes ago). The email body reads:  
Dear Investor,  
We are writing to inform you that the stock value of Infosys has exceeded the threshold you set in your notification preferences. The current value stands at 1434.5, surpassing your specified threshold of 1400.  
Given this development, we recommend reviewing your investment strategy and considering any necessary actions in accordance with your financial goals.  
Please let us know if you require further assistance or have any questions regarding this alert.

Below it, another message from the same sender is partially visible, sent at 17:36 (4 minutes ago). The bottom of the screen shows the browser's address bar with the full URL and the system tray with the date and time (22-04-2024, 17:41).

## VISUALIZATION:

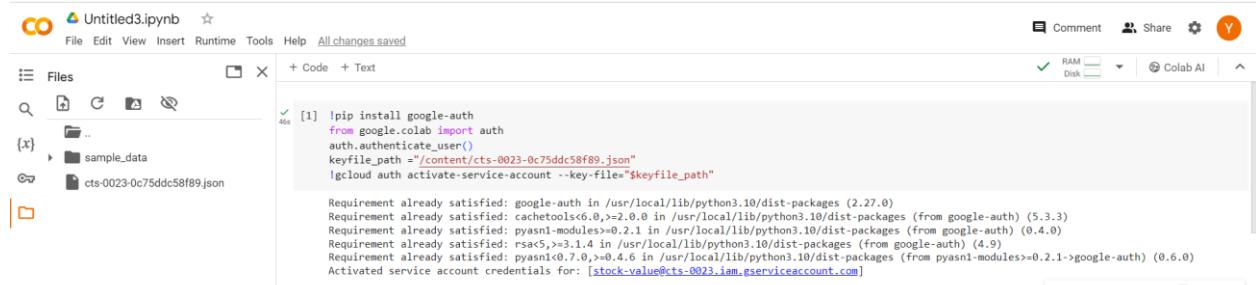
Here, we are creating a service account for authentication to access the cloud storage bucket.

The screenshot shows the Google Cloud IAM and admin interface. On the left, a sidebar lists various options under 'Service accounts'. The 'Service accounts' option is selected and highlighted in blue. In the main content area, a service account named 'stock-value' is being viewed. The 'DETAILS' tab is selected. The 'Service account details' section shows the name 'stock-value' and an email address 'stock-value@cts-0023.iam.gserviceaccount.com'. The 'Service account status' section shows the account is 'Enabled'. A 'DISABLE SERVICE ACCOUNT' button is visible. The top navigation bar includes the Google Cloud logo, project ID 'cts-0023', a search bar, and several icons.

create a json key file to service account.

The screenshot shows the 'KEYS' tab selected for the 'stock-value' service account. A warning message states: 'Service account keys could pose a security risk if compromised. We recommend that you avoid downloading service account keys and instead use the Workload Identity Federation. You can learn more about the best way to authenticate service accounts on Google Cloud here.' Below this, there is a note: 'Add a new key pair or upload a public key certificate from an existing key pair.' and 'Block service account key creation using organisation policies. Learn more about setting organisation policies for service accounts.' An 'ADD KEY' button is present. A table displays existing keys: one key is listed as 'Active' with the value '0c75ddc58f8965f92312a7e51cad01606b69ba87', created on '19 Apr 2024', and expiring on '1 Jan 10000'. The top navigation bar is identical to the previous screenshot.

Then, we'll store the json keyfile path in a variable for authentication.



A screenshot of the Google Colab interface. The top navigation bar shows "Untitled3.ipynb" and various menu options like File, Edit, View, Insert, Runtime, Tools, Help, and a "All changes saved" indicator. On the left, there's a sidebar titled "Files" with a tree view showing a folder structure: .., sample\_data, and a file named "cts-0023-0c75ddc58f89.json". The main workspace contains a single code cell with the following content:

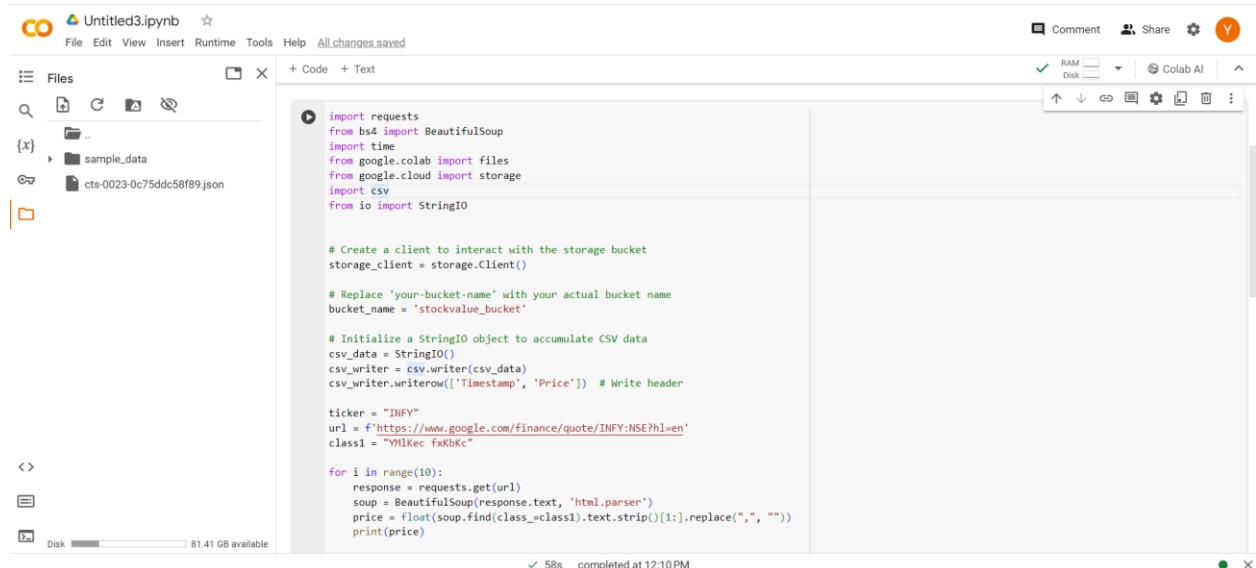
```
[1]: !pip install google-auth
      from google.colab import auth
      auth.authenticate_user()
      keyfile_path = "/content/cts-0023-0c75ddc58f89.json"
      !gcloud auth activate-service-account --key-file="$keyfile_path"

Requirement already satisfied: google-auth in /usr/local/lib/python3.10/dist-packages (2.27.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth) (4.9)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1>google-auth) (0.6.0)
Activated service account credentials for: [stock-value@cts-0023.jam.gserviceaccount.com]
```

The status bar at the bottom indicates "58s completed at 12:10PM".

We'll import csv module to store the stock price and its timestamp in a csv file.

Then, we created a storage client to create a bucket.



A screenshot of the Google Colab interface. The top navigation bar shows "Untitled3.ipynb" and various menu options like File, Edit, View, Insert, Runtime, Tools, Help, and a "All changes saved" indicator. On the left, there's a sidebar titled "Files" with a tree view showing a folder structure: .., sample\_data, and a file named "cts-0023-0c75ddc58f89.json". The main workspace contains a code cell with the following content:

```
import requests
from bs4 import BeautifulSoup
import time
from google.colab import files
from google.cloud import storage
import csv
from io import StringIO

# Create a client to interact with the storage bucket
storage_client = storage.Client()

# Replace 'your-bucket-name' with your actual bucket name
bucket_name = 'stockvalue_bucket'

# Initialize a StringIO object to accumulate CSV data
csv_data = StringIO()
csv_writer = csv.writer(csv_data)
csv_writer.writerow(['Timestamp', 'Price'])

# Initialize a StringIO object to accumulate CSV data
ticker = "INFY"
url = f"https://www.google.com/finance/quote/{ticker}:NSE?hl=en"
class1 = "MkIKe_fxKbkC"

for i in range(10):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    price = float(soup.find(class_=class1).text.strip()[1:].replace(",",""))
    print(price)

    # Write the current timestamp and price to the CSV
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
    csv_writer.writerow([timestamp, price])
```

The status bar at the bottom indicates "58s completed at 12:10PM".

We created a csv file with first 10 stock price values of INFY and stored in price\_output.csv

The screenshot shows a Google Colab notebook titled "Untitled3.ipynb". The code cell contains Python code for generating a CSV string from a dictionary, writing it to a blob in a Google Cloud Storage bucket, and then printing the uploaded file's URL. The output pane shows the printed URLs for each row of data, followed by a confirmation message: "CSV file uploaded to: gs://stockvalue\_bucket/price\_output.csv". The status bar at the bottom indicates the operation completed in 58 seconds at 12:10PM.

```
print(price)

# Write data to CSV string
csv_writer.writerow([time.strftime("%Y-%m-%d %H:%M:%S"), price])

time.sleep(5)

# Define the filename for the output CSV file
filename = 'price_output.csv'

# Upload the CSV data to Cloud Storage
bucket = storage_client.bucket(bucket_name)
blob = bucket.blob(filename)
blob.upload_from_string(csv_data.getvalue())

# Close the StringIO object
csv_data.close()

# Print the link to the uploaded CSV file
print(f'CSV file uploaded to: gs://{bucket_name}/{filename}')
```

1421.45  
1421.45  
1421.4  
1421.0  
1421.0  
1421.15  
1421.15  
1421.4  
1420.75  
CSV file uploaded to: gs://stockvalue\_bucket/price\_output.csv

✓ 58s completed at 12:10PM

The bucket created before adding object.

The screenshot shows the "Bucket details" page for the "stockvalue\_bucket" in the Google Cloud Storage console. The left sidebar shows "Cloud Storage" and "Buckets". The main area displays the "OBJECTS" tab of the "Folder browser", which is currently empty. The top navigation bar includes "GO TO PATH", "REFRESH", and "LEARN" buttons. The bottom of the page has filtering and sorting options for objects.

Cloud Storage    Bucket details    GO TO PATH    REFRESH    LEARN

Buckets > stockvalue\_bucket

UPLOAD FILES    UPLOAD FOLDER    CREATE FOLDER    TRANSFER DATA ▾  
MANAGE HOLDS    EDIT RETENTION    DOWNLOAD    DELETE

Filter by name prefix only ▾    Filter    Filter objects and folders    Show Live objects only ▾

Name	Size	Type	Created	Storage class	Last modified	Public access
No rows to display						

The object “price\_output.csv” is reflected in the bucket ‘stockvalue\_bucket’.

The screenshot shows the Google Cloud Storage interface. On the left, there's a sidebar with 'Cloud Storage' selected under 'Buckets'. The main area displays 'stockvalue\_bucket' details: Location (asia-south1 (Mumbai)), Storage class (Standard), Public access (Not public), and Protection (None). Below this, the 'OBJECTS' tab is selected in a navigation bar. A 'Folder browser' sidebar shows a single folder named 'stockvalue\_bucket'. The main content area lists objects in the 'stockvalue\_bucket' folder, including 'price\_output.csv' which has a size of 302 B, is a text/plain file, and was created on 22 Apr 2024, 12:10:22. There are also buttons for UPLOAD FILES, UPLOAD FOLDER, CREATE FOLDER, TRANSFER DATA, MANAGE HOLDS, EDIT RETENTION, DOWNLOAD, and DELETE.

We visualized data using Google Looker Studio by creating report.

The screenshot shows the Google Looker Studio interface. On the left, there's a sidebar with 'Recent' selected, showing options like 'Create', 'Shared with me', 'Owned by me', 'Bin', and 'Templates'. The main area has tabs for 'Recent', 'Reports', 'Data sources', and 'Explorer'. In the center, there's a large, empty workspace with a 'Create report' button at the bottom right. The top navigation bar includes 'Looker Studio', a search bar, and user account settings.

We extracted object from the google cloud storage bucket.

The screenshot shows the Google Data Studio interface with the title "Untitled Report". At the top, there's a toolbar with "Reset", "Share", "View", and other settings. Below the toolbar, a large button says "Add data to report". Underneath this button, there are two tabs: "Connect to data" (which is selected) and "My data sources". A search bar labeled "Search" is present. Below the search bar, there are four rows of data source cards. The fourth row contains the "Google Cloud Storage" card, which is highlighted with a blue border. Other cards include "Display & Video 360", "Extract Data", "Google Ad Manager", "Microsoft SQL Server", "MySQL", "NEW Search Ads 360", and "PostgreSQL". Each card has a brief description and a "By Google" note.

Finally, the report is created where y-axis represents Timestamp and x-axis represents Stock price.

The screenshot shows a completed Google Data Studio report titled "Untitled Report". The report features three main components: a table, a line chart, and a bar chart. On the left, there is a table titled "Timestamp" with 10 rows of data. The table includes columns for "Timestamp" and "Price". The data shows a sequence of timestamped stock prices. To the right of the table is a line chart titled "Record Count" with a single data series. The chart shows a sharp drop from a peak of 3 at 1421.4 to 1 at 1420.75. A callout box highlights the value "1420.75" and "Record Count: 1". Below the line chart is a bar chart showing the count of records for each price point. The x-axis lists prices: 1421.4, 1421.45, 1421.15, 1421, and 1420.75. The y-axis ranges from 0 to 5K. The bars show counts of approximately 4.5K for 1421.4, 3K for 1421.45, 3K for 1421.15, 2.8K for 1421, and 1.5K for 1420.75. The entire report is presented in a clean, organized layout with a light gray background.

Timestamp	Price
1. 22 Apr 2024, 06:40:23	1420.75
2. 22 Apr 2024, 06:40:17	1421.4
3. 22 Apr 2024, 06:40:12	1421.4
4. 22 Apr 2024, 06:40:06	1421.15
5. 22 Apr 2024, 06:40:00	1421.15
6. 22 Apr 2024, 06:39:55	1421
7. 22 Apr 2024, 06:39:49	1421
8. 22 Apr 2024, 06:39:43	1421.4
9. 22 Apr 2024, 06:39:38	1421.45
10. 22 Apr 2024, 06:39:32	1421.45

## 7.CONCLUSION

- The real-time stock price monitoring system successfully scrapes stock prices, publishes them to Pub/Sub, and triggers alerts. With the suggested enhancements, you can make it even more powerful and user-friendly.
- We built a real-time stock price monitoring system that uses Cloud Pub/Sub to stream stock price updates and Cloud Functions to analyze and send alerts.
- The system scrapes stock price data from a website, publishes it to a Pub/Sub topic, and triggers a Cloud Function.
- Scraping component scrapes stock price data from a specified URL using BeautifulSoup. We extract the stock price from the HTML page.
- Cloud Function component subscribes to the Pub/Sub topic. It analyzes the stock price data and sends alerts (email and SMS) if the stock price crosses a certain threshold.