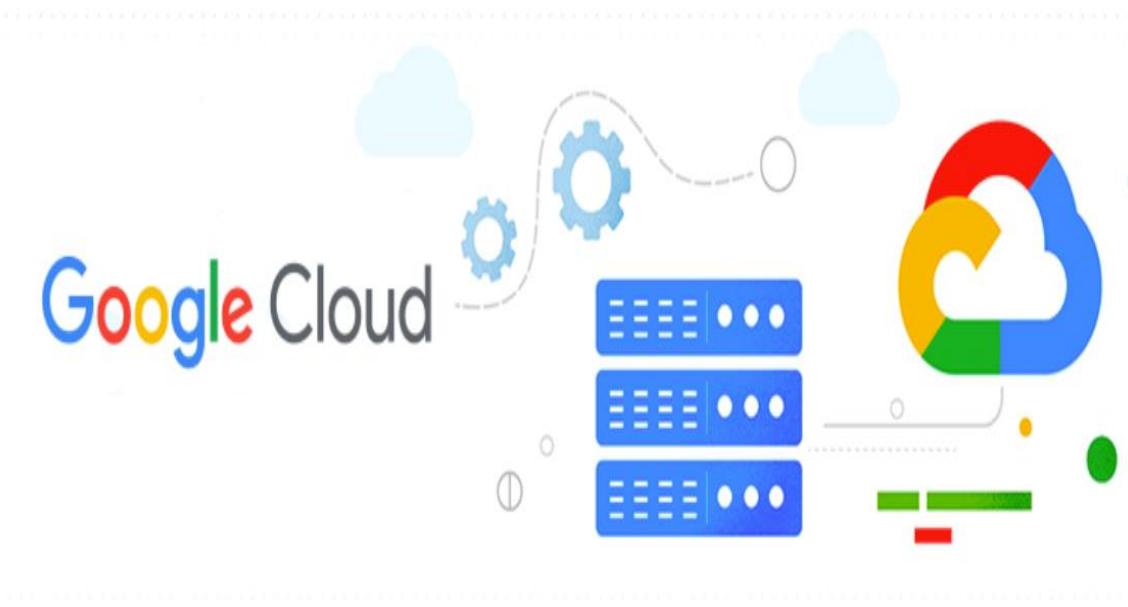


CREATE A CLOUD FUNCTION TO PULL A MESSAGE FROM PUB/SUB

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



Abhishek Maradi M

2320738

GenC Intern



CONTENTS

INTRODUCTION

OBJECTIVE

REAL TIME USECASE

SOFTWARE REQUIRED

PROJECT REQUIREMENT

IMPLEMENTATION

CONCLUSION

FUTURE ENHANCEMENTS

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.

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.



Here are some key points about GCP:

- 1. Infrastructure and 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.
3. **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.

REAL TIME USECASE:

Real-time Stock Price Monitoring

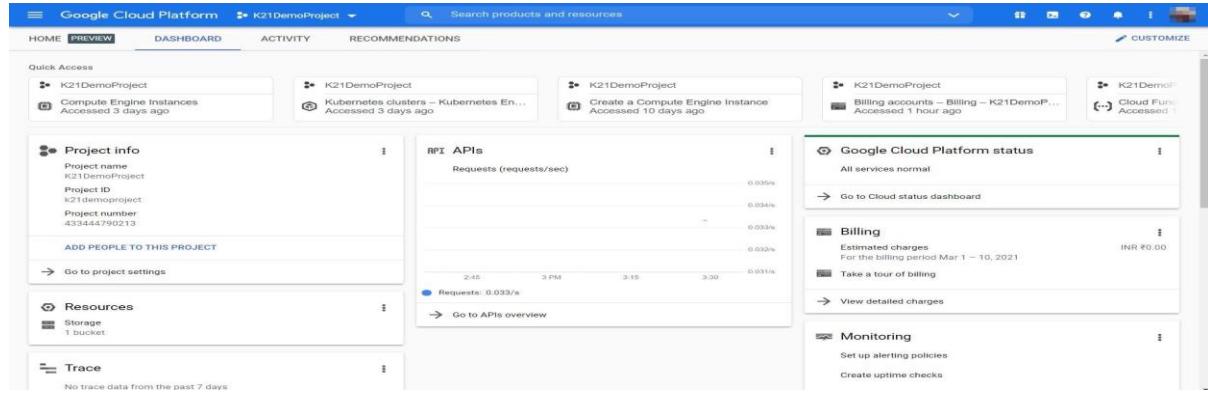
Build a real-time stock price monitoring system that alerts users when a stock price crosses a certain threshold. Use Cloud Pub/Sub to stream real-time stock price updates and Cloud Functions to analyse and send alerts.

Steps:

- Subscribe to a stock price API or feed and publish updates to Cloud Pub/Sub.
- Use Cloud Functions to analyse incoming price updates and check for threshold crossings.
- Send alerts via email, SMS, or push notifications using Cloud Functions.

SOFTWARE REQUIREMENTS :

GCP CONSOLE:



The first one is the Web Console. The moment you sign up with GCP, what you actually have is the web console and it is the front end, it is the gateway to dealing with a variety of services. As a beginner, you might want to spend more time with Web Console just getting yourself familiar with the navigation and understanding all the services offered by GCP.

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.

PROJECT REQUIREMENTS

Pub/Sub :



Cloud Pub/Sub

- Google Cloud Pub/Sub is a managed service to ingest data at scale.
- It is built using the publisher subscriber model where you have a set of publishers that send messages to a topic, and there are a set of subscribers that subscribe to the topic, and Pub/Sub provides the infrastructure for the publishers and subscribers to reliably exchange messages.
- Pub/Sub acts as a global entry point to GCP based analytics services. Whether you are ingesting elementary data logs or any of the data that is ingested into the Cloud. It is typically sent by a Cloud Pub/Sub.
- It acts as a reliable and simple staging location for data though Pub/Sub is not meant to be a durable data store. It can be used for staging data as it enters the Cloud and is waiting to get processed by either cloud dataflow or data proc.
- In fact, Pub/Sub can deliver the data to a variety of destinations depending on how the subscribers are configured. This is tightly integrated with services such as Cloud storage and Cloud Dataflow, where you can use Pub/Sub to store inbound data through - for real time processing through Dataflow or for historical dataset archival on Cloud storage.
- Cloud Pub/Sub supports at least once delivery with synchronous cross zone message replication. What this means is you actually get a highly reliable delivery mechanism based on Pub/Sub, and there is redundancy because of cross zone message replication.

- You don't lose messages when it is sent via Cloud Pub/Sub infrastructure. Unlike most of the services of Google Cloud platform comes with end to end encryption, integration with identity and access management, and audit logging.
- So all these capabilities give you additional mechanism to secure and also monitor the inbound data coming where Cloud Pub/Sub.

Cloud Functions:



**Google
Cloud
Functions**

- It's a serverless execution environment for building and connecting cloud services. So serverless computer environment is a mechanism where you don't have to provision and configure resources beforehand.
- This is fundamentally different from the way you deal with app engine, compute engine or even Kubernetes engine. For each of those you got to provision resources beforehand.
- We need to create app engine instances you need to launch VMs. We need to create a cluster, even before you can run your first line of code. But that's very different when it comes to cloud functions.
- In cloud functions, you write code as a function, which has a well-defined entry point and exit point, and you deploy that with no changes.
- That's the reason why it is called as a serverless compute environment where you don't have to provision a virtual machine or a container, to run your code. It typically, serverless compute environments respond to events.
- So instead of running this code forever, they get executed only when there is an external event. For example, adding a new object to a storage bucket or dropping a new message to the pub Sub queue, or for that matter, invoking a hit HTTP endpoint which will result in executing the serverless code.

- So any of those can be considered as the external event responsible for triggering the code. You can write cloud functions in JavaScript, Python 3, and Go. You don't have to package them in a specific format.
- At the most it's a zip file that gets into GCP environment and starts executing against events. GCP events fire a cloud function through a trigger. So trigger is what connects the external resource to a cloud function.

An example event could be adding a new object to a storage bucket, a classic use case in this scenario is converting high resolution images uploaded to Google Cloud storage to thumbnails. So every time a new high resolution image is added to a storage bucket, it triggers a cloud function and using an image manipulation library, it gets resized into a thumbnail and put in a different storage bucket. It this happens every time a new high risk image is added to the storage bucket. This is one of the most efficient and economical way of running code in the cloud. Triggers connect events to the function. So there is a trigger and there is an event and a function. So you define an event and you connect it to the function, and every time the trigger takes place it invokes the function via the event. So this is the environment that is very useful for executing code, which is written as code snippets or functions, and that's one of the reasons it's called as functions as a service or FaaS.

IMPLEMENTATION

Firstly, we created 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 is filled with 'final-demo'. Under 'Encryption', the 'Google-managed encryption key' option is selected. A 'CREATE' button is visible at the bottom. The left sidebar shows navigation options like 'Topics', 'Subscriptions', 'Snapshots', and 'Schemas' under 'Pub/Sub'.

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

The screenshot shows the 'final-demo' topic detail page. The 'Topic name' is listed as 'projects/cts-0023/topics/final-demo'. In the 'SUBSCRIPTIONS' section, a new subscription named 'final-demo-sub' is listed. A success message box at the bottom states: 'A new topic and a new subscription have been successfully created.' The left sidebar shows navigation options like 'Topics', 'Subscriptions', 'Snapshots', and 'Schemas' under 'Pub/Sub'.

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

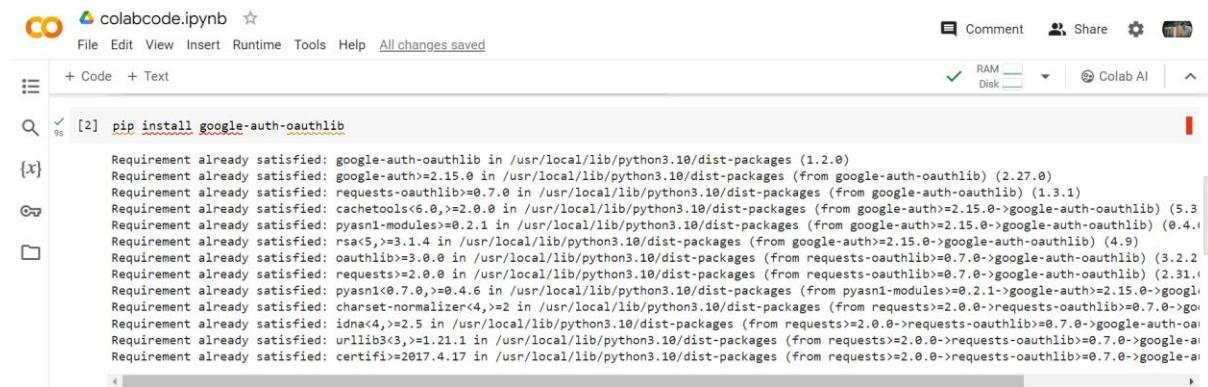
A screenshot of the Google Cloud Pub/Sub console. The left sidebar shows 'Topics' selected under 'Pub/Sub'. The main area displays a table of topics with one row highlighted: 'final-demo' (Topic ID), 'Google-managed' (Encryption key), 'projects/cts-0023/topics/final-demo' (Topic name), '-' (Retention), '-' (Ingestion source), and a three-dot menu icon. A success message box at the bottom center says 'A new topic and a new subscription have been successfully created.' The top navigation bar includes a warning message about payment changes for customers in India.

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

A screenshot of a Google Colab notebook titled 'colabcode.ipynb'. The code cell contains the command: [1] `pip install google.cloud.pubsub`. The output shows the package being downloaded and installed, with a progress bar indicating the download speed of 2.0 MB/s. The terminal output lists various dependencies and their versions, such as grpcio (~1.51.3), google-auth (~2.14.1), and requests (~2.18.0).

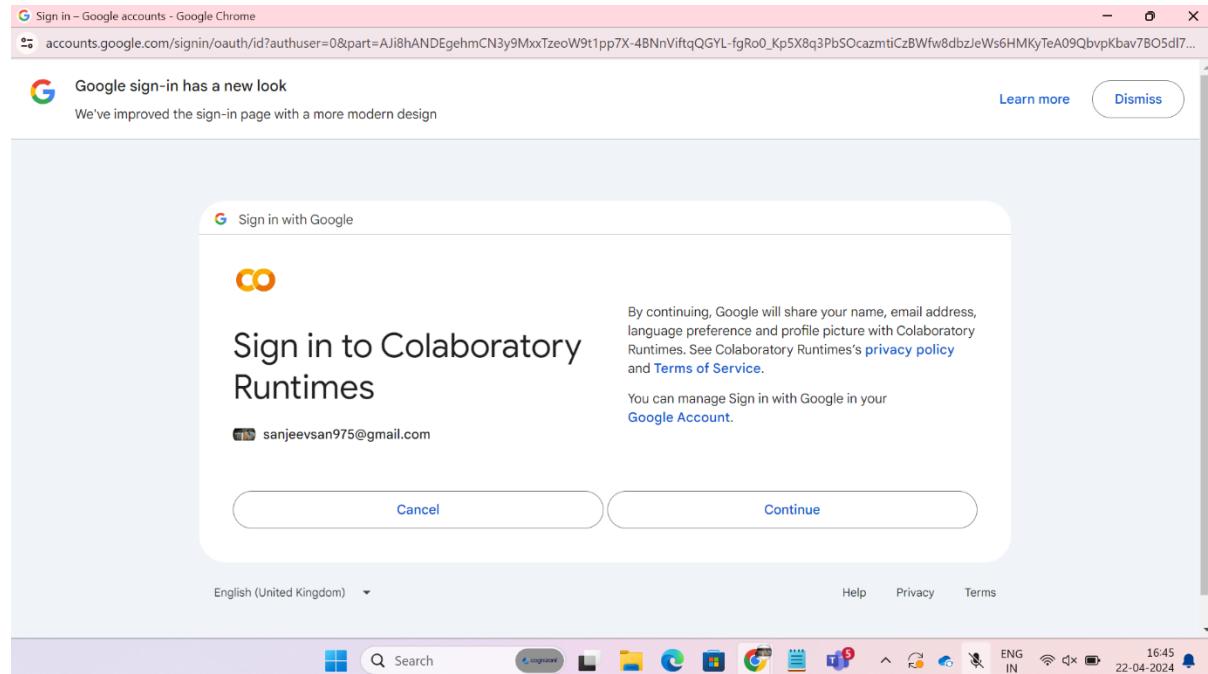
```
[1]: pip install google.cloud.pubsub
      Downloading google_cloud_pubsub-2.21.1-py2.py3-none-any.whl (273 kB)
      ...
Requirement already satisfied: grpcio<2.0dev,>=1.51.3 in /usr/local/lib/python3.10/dist-packages (from google.cloud.pubsub) (1.62.1)
Requirement already satisfied: google-auth<3.0.0dev,>=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.*,<3.0.0dev
Requirement already satisfied: proto-plus<2.0.0dev,>=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,!~4.21.4,!~4.21.5,<5.0.0dev,>=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<1.0.0dev,>=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.dev0,>=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.dev0,>=2.18.0 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]!~2.0.*)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.0dev,>=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<3.0.0dev,>=2.14.1>google.cloud.pubsub)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.0dev,>=2.14.1>google.cloud.pubsub) (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<3.0.0dev,>=2.14.1>google.cloud.pubsub)
Requirement already satisfied: charset-normalizer<4,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0>google-api-core)
Requirement already satisfied: idna<4,>=2.5.0 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0>google-api-core[grpc])
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=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.dev0,>=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.



```
colabcode.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[2] pip install google-auth-oauthlib
{x}
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>=2.15.0->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) (0.4.2)
Requirement already satisfied: charset-normalizer<4,>= in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib) (3.2.0)
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) (3.4.0)
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) (2.1.1)
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) (2023.10.1)
```

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()

import requests
from bs4 import BeautifulSoup
import time
import pandas as pd
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
```

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.



A screenshot of a code editor window showing Python code. The code is as follows:

```
5m  ✓
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
```

The code editor interface includes a toolbar at the top with icons for file operations, and a status bar at the bottom indicating the file is 5m 2s long and completed at 5:40PM.

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.



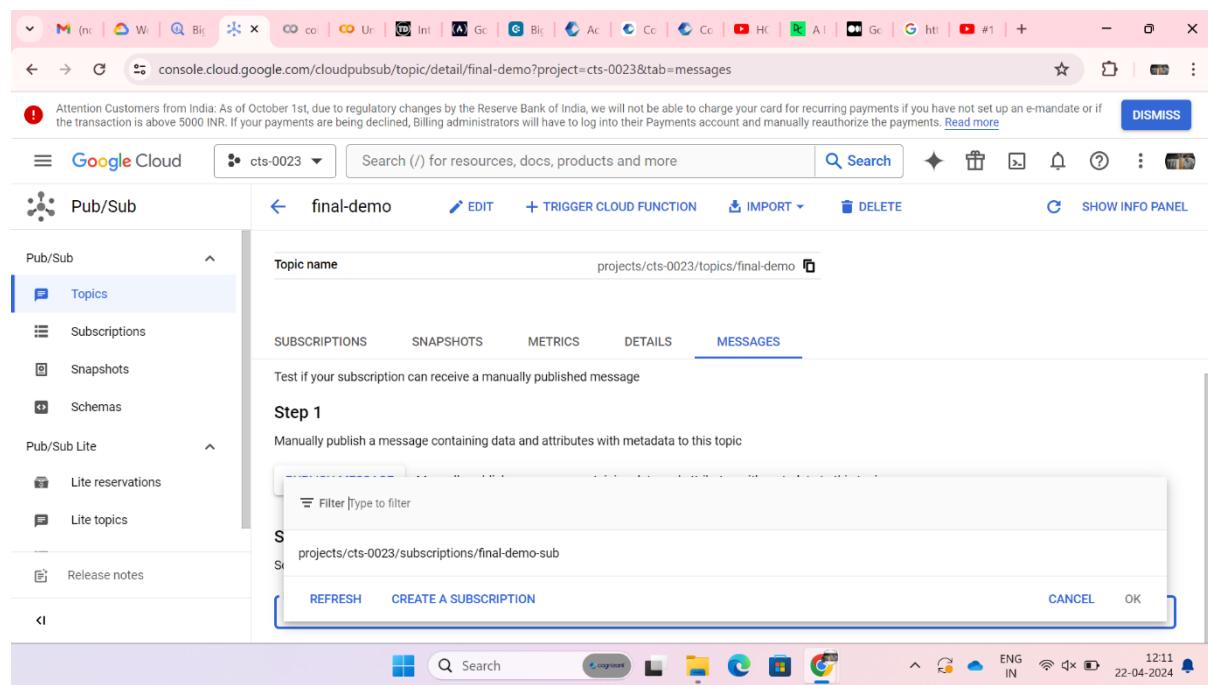
```
+ Code + Text
① # Main loop
for i in range(3):
    # Scrape price
    price = scrape_price(url, class_name)
    if price is not None:
        # Prepare message
        message = price

        # 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 has to be sent.



Attention Customers from India: As of October 1st, due to regulatory changes by the Reserve Bank of India, we will not be able to charge your card for recurring payments if you have not set up an e-mandate or if the transaction is above 5000 INR. If your payments are being declined, Billing administrators will have to log into their Payments account and manually reauthorize the payments. [Read more](#) DISMISS

Google Cloud cts-0023 Search (/) for resources, docs, products and more Search

Pub/Sub SHOW INFO PANEL

Topics EDIT + TRIGGER CLOUD FUNCTION IMPORT DELETE

Topic name projects/cts-0023/topics/final-demo

SUBSCRIPTIONS SNAPSHOTS METRICS DETAILS MESSAGES

Test if your subscription can receive a manually published message

Step 1
Manually publish a message containing data and attributes with metadata to this topic

Filter Type to filter

projects/cts-0023/subscriptions/final-demo-sub

REFRESH CREATE A SUBSCRIPTION CANCEL OK

The INFY stock price is published in topic messages.

This screenshot shows the Google Cloud Pub/Sub interface. On the left, a sidebar navigation bar includes 'Topics' (selected), 'Subscriptions', 'Schemas', and 'Pub/Sub Lite' sections. The main content area displays a 'Topic name' of 'final-demo' under 'projects/cts-0023/topics'. Below the topic name are tabs for 'SUBSCRIPTIONS', 'SNAPSHOTS', 'METRICS', 'DETAILS', and 'MESSAGES' (selected). A warning message states: 'Some messages or columns were truncated due to size. To pull the full message, see this documentation for an alternative approach.' The 'MESSAGES' table lists two entries:

| PUBLISH TIME | ATTRIBUTE KEYS | MESSAGE BODY | BODY | Ack |
|-----------------------|----------------|--------------|------|-----|
| 22 Apr 2024, 12:09:28 | - | 1421.45 | | ACK |
| 22 Apr 2024, 12:11:09 | - | 1420.5 | | ACK |

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

This screenshot shows the 'Create function' page in Google Cloud Functions. The 'Cloud Functions' tab is selected. The 'Basics' section includes fields for 'Environment' (2nd gen), 'Function name' (final-function), and 'Region' (asia-south1 (Mumbai)). The 'Trigger' section shows 'Trigger type' set to 'Cloud Pub/Sub' and 'Cloud Pub/Sub topic' set to 'projects/cts-0023/topics/final-demo'. At the bottom, there are 'NEXT' and 'CANCEL' buttons.

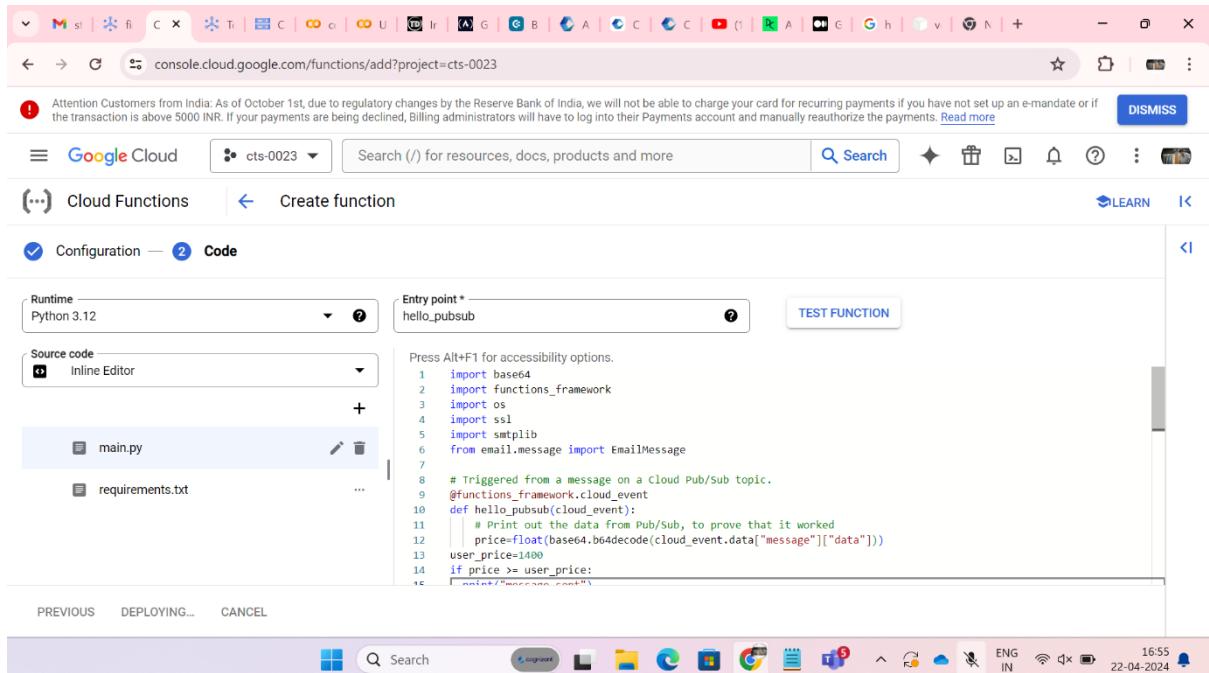
Here, we defined a cloud function .

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.



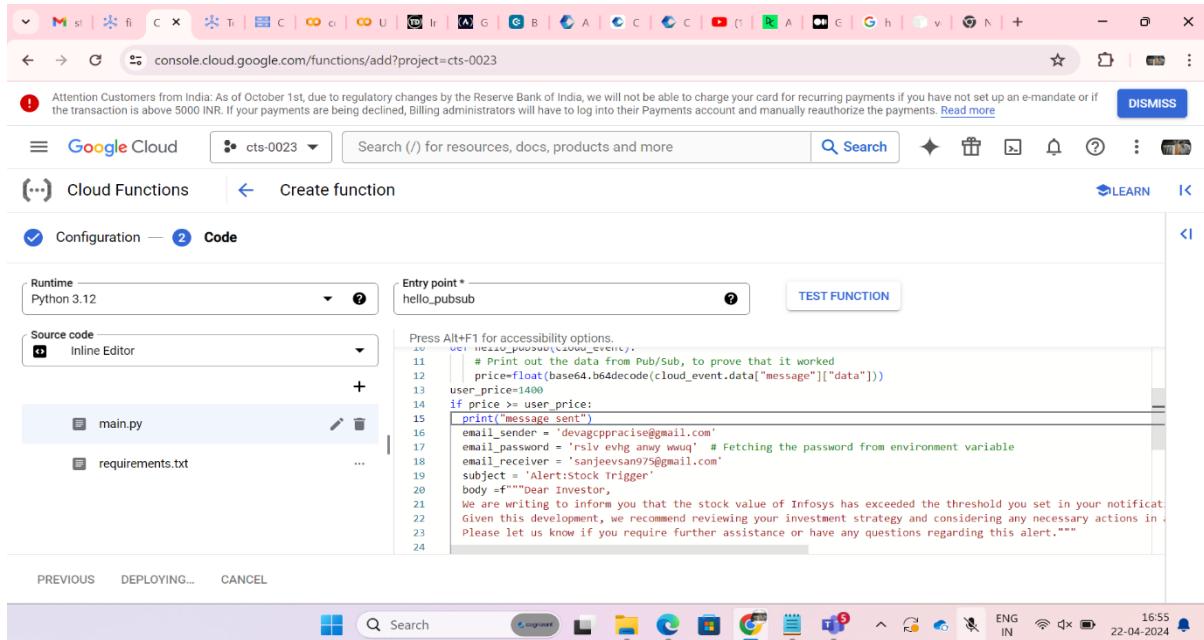
The screenshot shows the Google Cloud Functions deployment interface. The URL in the browser is `console.cloud.google.com/functions/add?project=cts-0023`. The page title is "Cloud Functions". The "Code" tab is selected. The runtime is set to "Python 3.12". The entry point is "hello_pubsub". The source code editor contains the following Python code:

```
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 of the code editor, there are buttons for "PREVIOUS", "DEPLOYING...", and "CANCEL". The status bar at the bottom right shows the time as 16:55 and the date as 22-04-2024.

The 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 configuration interface for a function named 'hello_pubsub'. The 'Code' tab is selected, displaying the following Python code in 'main.py':

```

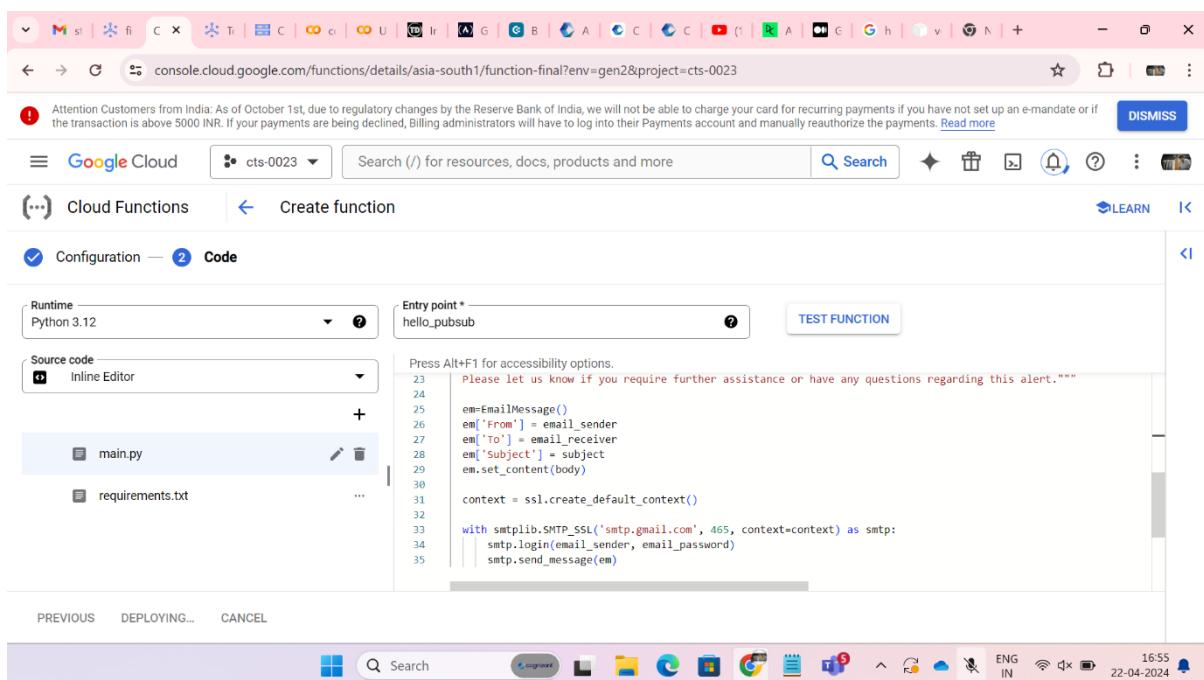
Runtime: Python 3.12
Entry point: hello_pubsub
Source code: Inline Editor
main.py
requirements.txt

Press Alt+F1 for accessibility options.
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 sent")
16         email_sender = 'devagcppractice@gmail.com'
17         email_password = 'rslv evhg anvy wuug' # Fetching the password from environment variable
18         email_receiver = 'sanjeevanan97@gmail.com'
19         subject = 'Alert:stock Trigger'
20         body = """Dear Investor,
21             We are writing to inform you that the stock value of Infosys has exceeded the threshold you set in your notification.
22             Given this development, we recommend reviewing your investment strategy and considering any necessary actions in this regard.
23             Please let us know if you require further assistance or have any questions regarding this alert."""
24

```

The code sends an email to 'sanjeevanan97@gmail.com' using a Gmail account with password 'rslv evhg anvy wuug'. The subject is 'Alert:stock Trigger' and the body contains a message about a stock value exceeding a threshold.

An instance of `EMAILMESSAGE` class is created and that instance is used to create from, to, subject and body fields of email.



The screenshot shows the Google Cloud Functions configuration interface for the same function 'hello_pubsub'. The 'Code' tab is selected, displaying the completed Python code in 'main.py':

```

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

Press Alt+F1 for accessibility options.
23     Please let us know if you require further assistance or have any questions regarding this alert."""
24
25     em=EmailMessage()
26     em['From'] = email_sender
27     em['To'] = email_receiver
28     em['Subject'] = subject
29     em.set_content(body)
30
31     context = ssl.create_default_context()
32
33     with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
34         smtp.login(email_sender, email_password)
35         smtp.send_message(em)

```

The code uses the `EmailMessage` class to create an email message with 'email_sender' as the 'From' field, 'email_receiver' as the 'To' field, 'subject' as the subject, and 'body' as the content. It then connects to a Gmail SMTP server at port 465 using SSL/TLS and logs in with the provided credentials to send the message.

The cloud function got deployed.

A screenshot of the Google Cloud Functions details page. At the top, it shows the URL: console.cloud.google.com/functions/details/asia-south1/function-final?env=gen2&project=cts-0023. A message at the top states: "Attention Customers from India: As of October 1st, due to regulatory changes by the Reserve Bank of India, we will not be able to charge your card for recurring payments if you have not set up an e-mandate or if the transaction is above 5000 INR. If your payments are being declined, Billing administrators will have to log into their Payments account and manually reauthorize the payments. [Read more](#)". A "DISMISS" button is present. Below the message, the function name is "function-final" (2nd gen), deployed at 22 Apr 2024, 17:08:44, with a URL: <https://asia-south1-cts-0023.cloudfunctions.net/function-final>. A "Powered by Cloud Run" badge is shown. The "METRICS" tab is selected, displaying two charts: "Invocations/Second" and "Execution time". The "Invocations/Second" chart shows a single data point at 0.015/s. The "Execution time" chart shows a single data point at 10ms. The bottom of the screen shows the Windows taskbar with various icons and the date/time: 22-04-2024, 17:08.

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

A screenshot of the Gmail inbox. The subject of the top email is "Alert:Stock Trigger" and it is from "devagcppraccise@gmail.com" (to me). 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.

The bottom of the screen shows the Windows taskbar with various icons and the date/time: 22-04-2024, 17:41.

VISUALISATION:

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

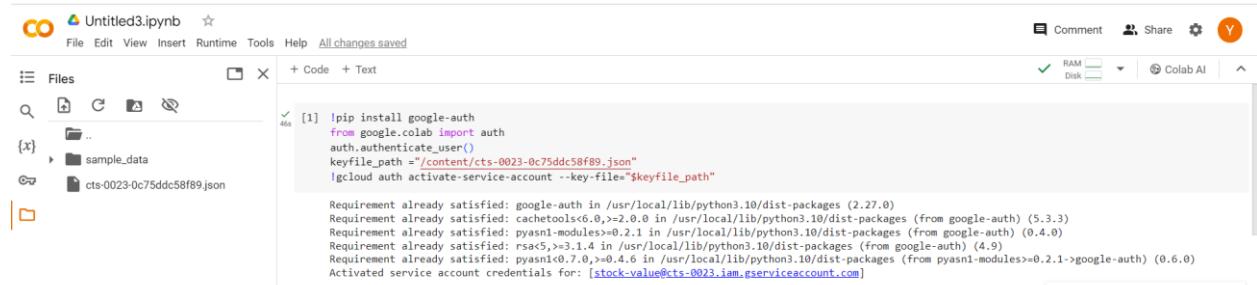
The screenshot shows the Google Cloud IAM and admin interface. On the left, a sidebar lists various services: IAM, Identity and organisation, Policy troubleshooter, Policy analyser (NEW), Organisation policies, Service accounts (which is selected), Workload Identity Federat..., and Workforce identity federat... Below the sidebar, a main panel displays the details of a service account named 'stock-value'. The 'DETAILS' tab is active. 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 at the bottom of this section.

We created a json key file to service account.

The screenshot shows the Google Cloud IAM and admin interface, specifically the 'KEYS' tab 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, instructions say '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.' A 'ADD KEY' button is present. A table below lists a single key entry:

| Type | Status | Key | Key creation date | Key expiry date |
|------------------|--------|--|-------------------|-----------------|
| Google Cloud Key | Active | 0c75ddc58f8965f92312a7e51cad01606b69ba87 | 19 Apr 2024 | 1 Jan 10000 |

Then, we stored the json keyfile path in a variable for authentication.

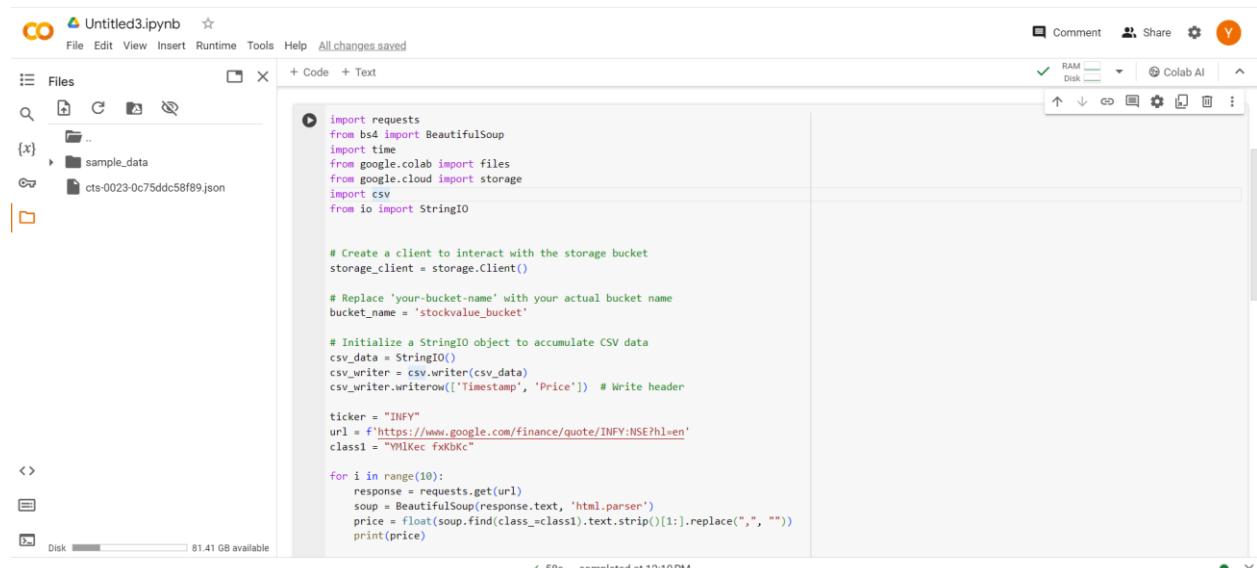


```
[1] !pip install google-auth
from google.colab import auth
auth.authenticate_user()
keyfile_path = "/content/cts-0023-0c75dd58f89.json"
!gcloud auth activate-service-account --key-file="$keyfile_path"

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@ts-0023.jam.gserviceaccount.com]
```

We imported csv module to store the stock price and its timestamp in a csv file.

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



```
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'])

ticker = "INFY"
url = f"https://www.google.com/finance/quote/{ticker}:NSE?hl=en"
class1 = "MNIKec 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 data to CSV
    csv_writer.writerow([time.time(), price])
```

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 interface. A code cell contains Python code for generating a CSV file and uploading it to a Cloud Storage bucket. The output of the code shows the uploaded file's contents and its location in the bucket.

```
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())

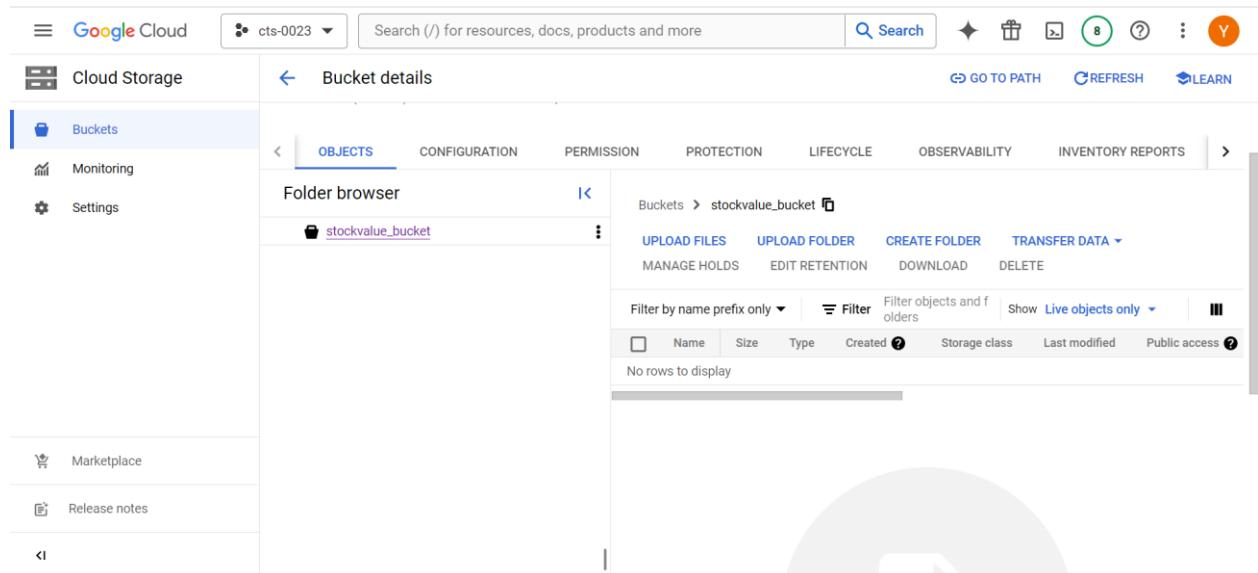
# Close the StringIO object
csv_data.close()

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

Output:

```
1421.45
1421.45
1421.4
1421.4
1421.0
1421.0
1421.15
1421.15
1421.4
1421.4
1420.75
CSV file uploaded to: gs://stockvalue_bucket/price_output.csv
```

The bucket created before adding object.



The screenshot shows the Google Cloud Storage console. The left sidebar is collapsed. The main area displays the 'Bucket details' page for 'stockvalue_bucket'. The 'OBJECTS' tab is selected, showing a 'Folder browser' with a single entry: 'stockvalue_bucket'. Below the browser are buttons for 'UPLOAD FILES', 'UPLOAD FOLDER', 'CREATE FOLDER', and 'TRANSFER DATA'. There are also buttons for 'MANAGE HOLDS', 'EDIT RETENTION', 'DOWNLOAD', and 'DELETE'. A filter bar at the bottom allows filtering by name prefix, type, and other parameters. A message at the bottom states '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, a sidebar has 'Buckets' selected. The main area shows 'stockvalue_bucket' details: Location (asia-south1 (Mumbai)), Storage class (Standard), Public access (Not public), and Protection (None). Below this is a tabbed navigation bar with 'OBJECTS' selected. The 'Folder browser' section shows a list of objects under 'stockvalue_bucket'. One object, 'price_output.csv', is listed with the following details:

| Name | Size | Type | Created |
|------------------|-------|------------|-----------------------|
| price_output.csv | 302 B | text/plain | 22 Apr 2024, 12:10:12 |

We visualized data using Google Looker Studio by creating report.

The screenshot shows the Google Looker Studio interface. The top navigation bar includes 'Looker Studio', a search bar, and user account options. The main area features a sidebar with 'Recent' (selected), 'Shared with me', 'Owned by me', 'Bin', and 'Templates'. A central workspace contains a large 'Create' button with a grid icon. At the bottom, there's a 'Create report' button.

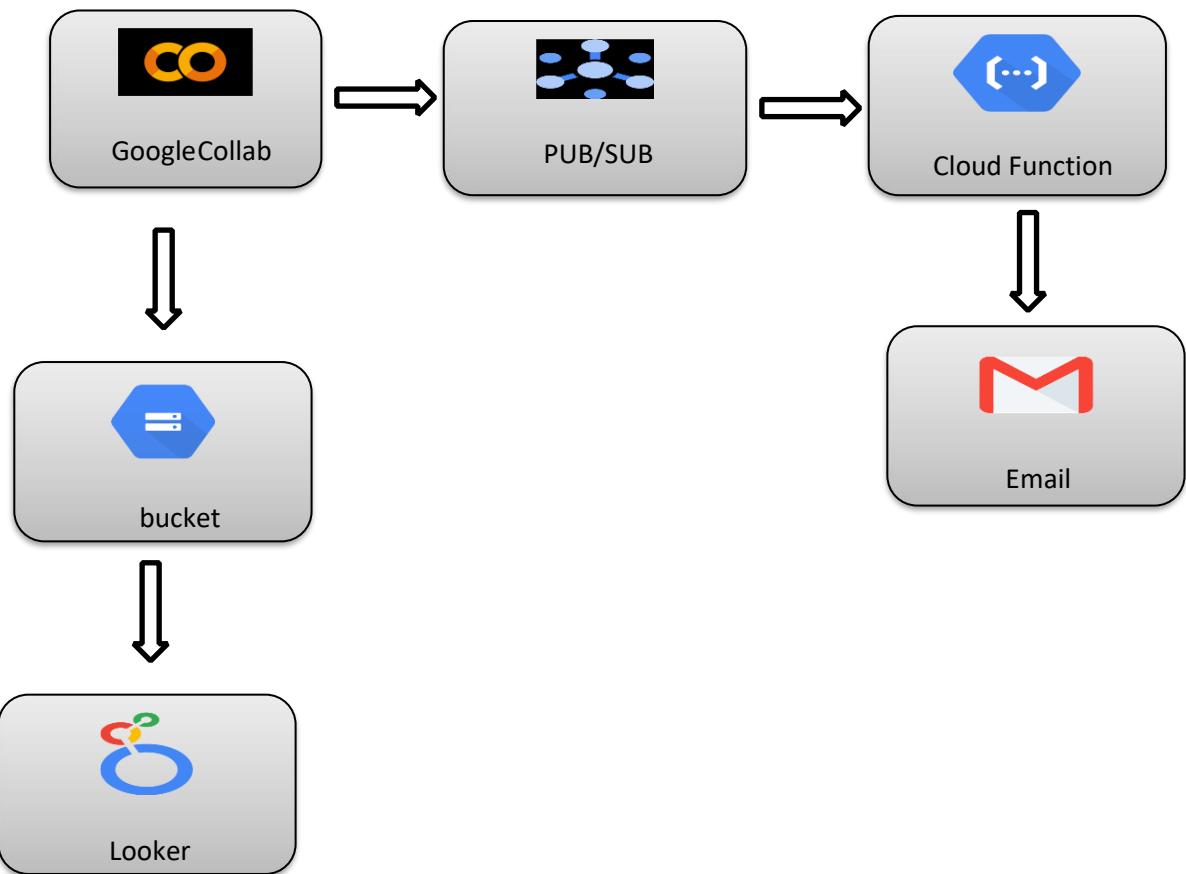
We extracted object from the google cloud storage bucket.

The screenshot shows the 'Add data to report' screen in Google Data Studio. At the top, there's a toolbar with 'Untitled Report' and various icons for file operations, sharing, and viewing. Below the toolbar, a search bar labeled 'Search' is followed by a grid of data source cards. The cards are organized into four main sections: Google, Google, Microsoft, and MySQL. The 'Google' section contains cards for Apigee, Campaign Manager 360, Cloud Spanner, and Google Cloud SQL for MySQL. The second 'Google' section contains cards for Display & Video 360, Extract Data, Google Ad Manager, and Google Cloud Storage. The Microsoft section contains a card for Microsoft SQL Server. The MySQL section contains a card for MySQL. The 'Google Cloud Storage' card is specifically highlighted with a blue border, indicating it has been selected or is the current focus.

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 includes three main components: a table, a line chart, and a bar chart. The table on the left lists 10 timestamp entries with their corresponding prices. The line chart in the center plots 'Record Count' against 'Timestamp', showing a sharp decline from 3 to 1 between 1421.4 and 1420.75. The bar chart at the bottom shows the distribution of stock prices, with the highest frequency at 1421.4. The interface includes standard Data Studio navigation and sharing tools at the top.

| Timestamp | Price |
|-----------------------|---------|
| 22 Apr 2024, 06:40:23 | 1420.75 |
| 22 Apr 2024, 06:40:17 | 1421.4 |
| 22 Apr 2024, 06:40:12 | 1421.4 |
| 22 Apr 2024, 06:40:06 | 1421.15 |
| 22 Apr 2024, 06:40:00 | 1421.15 |
| 22 Apr 2024, 06:39:55 | 1421 |
| 22 Apr 2024, 06:39:49 | 1421 |
| 22 Apr 2024, 06:39:43 | 1421.4 |
| 22 Apr 2024, 06:39:38 | 1421.45 |
| 22 Apr 2024, 06:39:32 | 1421.45 |



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.
- Pub/Sub publishes the scraped stock price data to a Pub/Sub topic.

- 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.

FUTURE ENHANCEMENTS

Customizable Alerts:

- Allow users to set personalized alert thresholds for each stock.
- Implement different notification channels (e.g., email, SMS, push notifications) based on user preferences.

Machine Learning Predictions:

- Train ML models to predict stock price movements.
- Use historical data to provide insights and recommendations.

Interactive Dashboard:

- Create a web-based dashboard to visualize real-time stock prices.
- Include charts, graphs, and historical data.

Mobile App Integration:

- Develop a mobile app for users to receive alerts on their smartphones.
- Provide a seamless user experience.

Market News Integration:

- Fetch relevant news articles related to monitored stocks.
- Display news alongside stock prices.