DataEng S24: Project Assignment 1

Gather and Transport

Due date: Apr 21, 2024 at 10pm

The first step in building your data pipeline is to provide mechanisms for gathering and transporting the raw data. We have developed a simple web server that provides access to one day's worth of TriMet GPS 5-second-interval "breadcrumb" data for the buses in the TriMet system.

Here's how you can access the data:

1. Obtaining Vehicle IDs:

- You will be assigned a list of vehicle IDs shortly after the project teams are finalized.
- Your assigned list of vehicle IDs can be found here.

2. Accessing Breadcrumb Data:

- Use the following URL to access the breadcrumb data

```
https://busdata.cs.pdx.edu/api/getBreadCrumbs?vehicle_id=<vehicle_id>
```

- Make sure to replace <vehicle_id> with the specific bus ID you wish to retrieve data for. The ID's will be provided to you in your assigned list of vehicle IDs.

Your job is to gather and transport this data once per day. At first you can just use a web browser to download the data, and save it to a file, but by the assignment due date you will:

A. Create, configure and use a Google Cloud Platform (GCP) linux virtual machine (VM).
B. Develop a simple python program to gather the data programmatically.
C. Configure your VM running your gathering client to run daily.
D. Allocate and configure a message passing "topic" and "subscription" at Google Cloud Pub/Sub.
E. Enhance your data gathering client to parse the breadcrumb data and publish individual JSON records.
F. Enhance your data gathering client to send the individual breadcrumb records to your Cloud Pub/Sub topic.
G. Develop a python program to receive the breadcrumb readings from the Pub/Sub topic and save them to file, one file per day.
H. Configure your VM to run your Pub/Sub receiver constantly so that it always receives all new data.
I. Schedule your VM to start and stop automatically.

Before you begin, make sure that you can access:

```
https://busdata.cs.pdx.edu/api/getBreadCrumbs?vehicle_id=<vehicle_id>
```

Use any vehicle id assigned to you as part of the project and study the data that it provides. It provides JSON data containing one JSON object per breadcrumb sensor reading. Examine the attributes, and ask questions on the class slack channel if you have any questions about the data.

You can only download data for the current day and not for any past or future days. You will have a 24-hour window to download the data for that day. This website uses the Pacific Time Zone.

Extra Credit

This project only requires one VM (e2-medium type VM). However, GCP allows you to create multiple Compute Engine VM's (Virtual Machines), so you can make the project more interesting and realistic (and earn extra credit in the process) if you utilize separate VMs for your data publisher and receiver clients.

A. Create, Configure and Run Your Virtual Machine

We provided you with information about how to obtain your GCP credits. Use them well and monitor your credit availability throughout the quarter. If you have never used GCP before, then this is a good time to learn. Try one of the free, online "How to Use GCP" tutorials available on the internet. Also, see: Create a Linux VM instance in Compute Engine. Note: primarily we plan to use the "Compute Engine" features of GCP, so for now you can ignore mention of such things as "Kubernetes Engine", "App Engine", "SQL BigData", and so on.

For this class, create a new project in your GCP account. You can do so from your dashboard (Google Cloud Platform) on the top left navigation bar. **Make sure your Virtual Machine is running on Pacific time zone** so that your VM's timestamps will make sense for you and for your data. Do all of your class-related tasks within this project.

When redeeming GCP credits and in general when using GCP, **make sure you are logged out of all other Google accounts other than your @pdx.edu school account.** Being logged into multiple Google accounts is the #1 cause of configuration errors with GCP-related school projects. We suggest you open a new incognito window in Chrome (private window in Firefox), log into your PSU Google account and then redeem your credits and use cloud.google.com (GCP).

To create a VM, go to your GCP Console. That is, in a browser go to console.cloud.google.com, Sign In with your @pdx.edu student email account address if necessary. From the Navigation menu (open with the hamburger menu on the LH side of the screen) select Compute Engine > VM Instances or you can search for "Compute Engine" in the search bar. If you're taken to a product details page to enable the Compute Engine API, please select enable.

Select “Create Instance”. This will take you to a screen that allows you to configure the properties of your VM Instance before you actually create it. For the most part, you can just leave the configuration parameters as-is and take the defaults, but you should modify these parameters:

- Select the region as “us-west1(Oregon)”
- Machine Type: choose a “e2-medium” machine type

Select the “Create” button at the bottom to create your instance. This will take you to a VM Instances management page. After a few moments you should see that your VM is up and running.

SSH to your new VM. You can use the GCP Console by clicking on the SSH button, or you can utilize the command-line interface (CLI). For Cloud Shell, execute the following command.

```
gcloud compute ssh [INSTANCE_NAME] --zone [ZONE]
```

Make sure that you can access the breadcrumb server by running a command line client such as curl, like this:

```
curl -L https://busdata.cs.pdx.edu/api/getBreadCrumbs?vehicle_id=<vehicle_id> --output <output_file_name>
```

Again, make sure to pass the vehicle id assigned to you. See the man page for curl as it can be a very useful tool for testing connectivity. The ‘-L’ flag instructs curl to follow any server redirects.

B. Initial Python Data Gatherer

Your task is to create a Python program capable of making an HTTP request to a web server and fetching the results. While numerous Python packages handle HTTP requests, we recommend utilizing the well-established "urllib" Python package. Alternatively, you may choose the "requests" library, which provides similar functionality. Either way, ensure the package is installed on your VM.

If you have a multi-person team, then consider assigning one team member to handle the creation and debugging of this python program.

Initially, the program's primary function is to write the data to a file and then terminate. In Step E, you will improve the program by parsing the JSON data and sending individual records to a Pub/Sub topic.

C. Run the Data Gatherer Daily

On a linux system there are many ways to run programs automatically, periodically. We suggest that you investigate a simple option such as cron. See this tutorial which explains the basics of using cron on Ubuntu linux: How To Use Cron to Automate Tasks on Ubuntu 18.04. Here’s another good resource to learn about cron: Crontab.guru.

Your job is to use cron (or some other mechanism of your choice) to run your python data gatherer daily. You can run it at any time of day. We suggest that during your debugging you configure cron to run your client more frequently so that both cron and your python program are working correctly. When you are happy with the results then re-configure cron to run once or twice per day.

Test: After you are comfortable that cron (or your favored mechanism) is configured properly, then wait one day and make sure that your data is being gathered automatically as expected. Success? Congratulations! You successfully built a data pipeline, an automated software system that moves a continuous, unbounded source of data from an initial state closer to its destination state. It might not be complete, but it's a good start.

Extra Credit: Make your data pipeline reliable and tolerant of VM crashes, network failures, limited storage space and other types of unexpected computer systems shortages and problems. We do not expect your pipeline to be ultra-reliable, but it can be fun to do.

```
snutheti@instance-20240415-000904:~/data$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
35 11  * * * /home/snutheti/data_gatherer.sh >> /home/snutheti/cron.log 2>&1
```

D. Configure Google Cloud Pub/Sub

Pub/Sub is a fully managed messaging service provided by Google Cloud Platform, designed to facilitate asynchronous communication between independent applications and systems. It serves as a reliable, scalable, and low-latency platform for handling real-time data feeds and event-driven workflows in cloud-native architectures.

Your pipeline will be relatively small with well-known publishers and receivers, so it probably does not need something like Pub/Sub, but we will use it anyway so that you get some valuable experience with general publisher/receiver stream-oriented systems. You will use Google Cloud Pub/Sub, which offers

seamless integration with other GCP services and simplifies the development of distributed applications and real-time data processing pipelines in cloud environments.

Complete the steps in the separate project instruction document for the Pub/Sub configuration: link
If you are doing this project as a multi-person team, then consider assigning one teammate to handle all Pub/Sub-related configuration for your project. Also, we will be using Pub/Sub in class in week 3 (link), so all members of your team will get some hands-on experience with it.

E. Parse JSON into Individual Breadcrumb Records

Update your Data Publisher python program to fetch the data from the URL and parse the JSON data received from the breadcrumbs service.There are many ways to parse JSON data in python, we recommend using the basic, built-in json module provided in python.

Handling the breadcrumb data as individual records might not be the most efficient way to transport the data, it might be more efficient to collect the breadcrumb readings into larger aggregates such as “trip”, “route”, “bus” or “hour of the day”. But we require you to send each reading individually to get a better understanding of how the data pipeline might work in a live system in which buses emit GPS locations dynamically throughout the day.

F. Send Breadcrumb Records to Pub/Sub Topic

Create a new Pub/Sub topic to use for your breadcrumbs sensor readings if you haven't created one previously in step D.

Modify your Data Publisher python code to send each sensor reading as a separate message to the Pub/Sub topic.

G. Pub/Sub Receiver

Modify the receiver.py program used in the Pub/Sub tutorial (week 3) to receive the data from your new Pub/Sub topic. Your receiver program should write all received data to file, one file per day. At this point the receiver's primary task is to simply write the incoming records to a file. You will enhance this program in the future to do validation, transformation, enhancement and storage of the sensor readings.

H. Configure Linux to Run Pub/Sub Receiver Continually

To keep the pipeline running, you need to make sure that your Pub/Sub receiver runs indefinitely. There are many ways to keep a process running on Linux, but here we are suggesting that you configure your

program as a systemd service. See this tutorial for more information about how to configure a systemd service: [Setup a python script as a service through systemctl/systemd | by WasiUllah Khan].

Once you follow the steps in the tutorial above, you can test the output continually with the following commands:

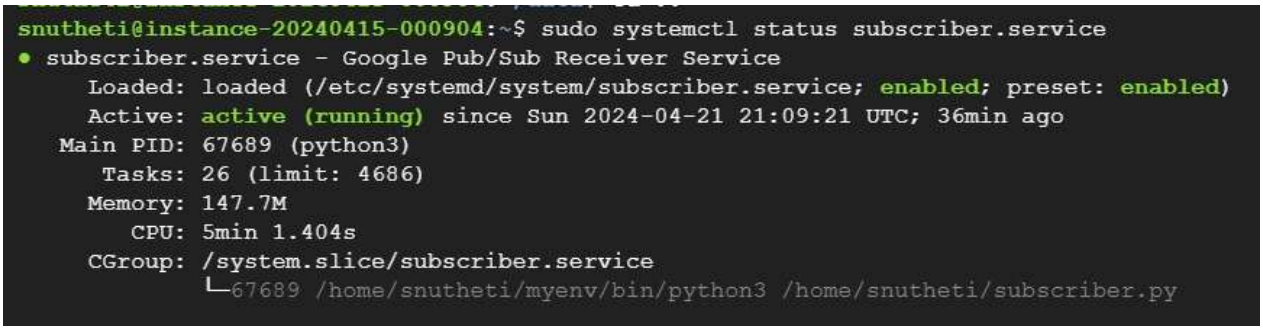
tail -f timestamp.txt

or

tail -f /timestamp.txt

If you notice that the timestamp is "incorrect", it might be because your VM is not set to Pacific Standard Time. See if you can fix it by reading through this: [Google Compute Engine time zone - Stack Overflow].

If you have a multi-person team, then consider assigning this as a separate concurrent task because it can be done separately long before the completion of other tasks in this project assignment.

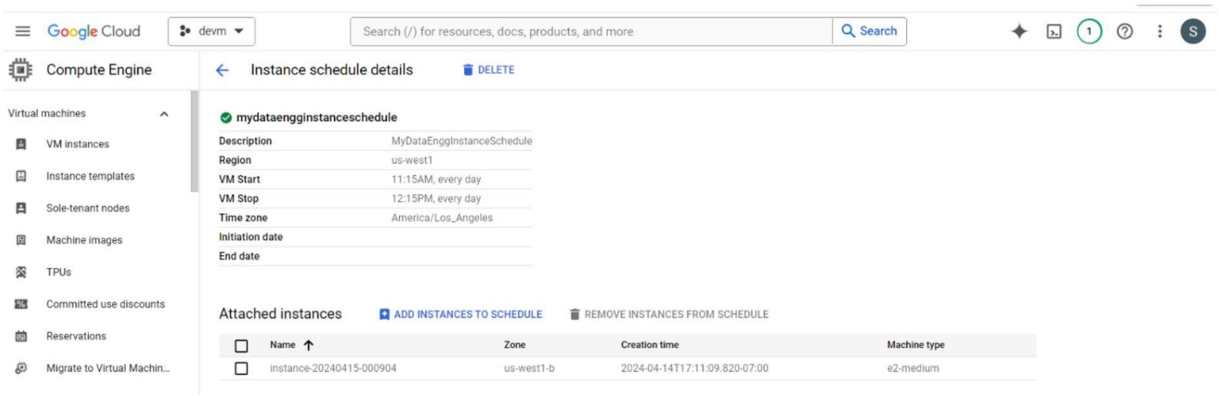


I. Schedule your VM to start and stop automatically

Now that you have a publishing client and a receiving client that runs automatically to do their respective jobs, you might realize that your VM has to be running for these clients to run as well.

Keeping your VM on continuously is a sure way to run out of GCP credits quickly, but it's a hassle to manually go to the GCP console to turn it on and off everyday. If you're manually doing that, what's the point of all your previous work to automate with cron and systemd? Not to mention that manual work leaves room for human errors, e.g. what if you forget one day or more days; what if you don't have access to your computer for a while (heaven forbids something were to happen to a student's computer!); or what if you're out of commission due to illness or sickness.

Check out this page for help with setting up a VM instance schedule: [Scheduling a VM instance to start and stop | Compute Engine Documentation | Google Cloud]. Incorporate this into your project, so you can eliminate the manual work and enjoy the benefits of your earlier cron and systemd setups.



Submission

Congratulations! You now have a working data pipeline consisting of multiple python programs connected with an event stream. It runs automatically and is ready to be enhanced with data transformations and loading to a database server (Assignment 2).

To submit your completed Assignment 1, create a Google document containing the table shown below. Share the document as viewable by anybody at PSU who has the link. You do NOT need to share it with the individual instructors. Then include the URL of the document in the DataEng Project Assignment Submission form.

Your Code

Provide a reference to the GitHub repository where you store your python code. If you are keeping it private then share it with the Professor (rbi@pdx.edu or mina8@pdx.edu) and TA (vysali@pdx.edu).

—

DataEng Project Assignment 1 Submission Document

Construct a table showing each day for which your pipeline successfully, automatically processed one complete days' worth of sensor readings.

Date	Day of Week	Approximate Time of day for your data access	# Sensor Readings	Total Data Saved (KBs)	# Pub/Sub messages published and received
04-19-2024	Friday	11:35 AM PDT	{"EVENT_NO_TRIP": 223142392, "EVENT_NO_STOP": 223142402, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3150, "METERS": 309807, "ACT_TIME": 80155, "GPS_LONGITUDE": -122.563325, "GPS_LATITUDE> : 45.495197, "GPS_SATELLITES": 11.0, "GPS_HDOP": 1.2)	80316	335570
04-20-2024	Saturday	11:35 AM PDT	("EVENT_NO_TRIP": 223949278, "EVENT_NO_STOP": 223949299, "OPD_DATE": "21DEC2022:00:00:001 "VEHICLE_ID": 3206, "METERS": 167041, "ACT_TIME": 56244, "GPS_LONGITUDE" -122.74932, "GPS_LATITUDE": 45.486008, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.6)	68796	287592
04-21-2024	Sunday	11:35 AM PDT	“EVENT_NO_TRIP": 224552413, "EVENT_NO_STOP": 224552515, "OPD_DATE": "22DEC2022:00:00:00", "VEHICLE_ID": 3901, "METERS": 185292, "ACT_TIME": 49491, "GPS_LONGITUDE": -122.52211, "GPS_LATITUDEITUDE": 45.519047, "GPS_SATELLITES": 12.0,	77744	324935

			"GPS_HDOP": 0.77		

Additionally, include screenshots for the parts C, H and I

1. Output of crontab -l: Your scheduled cron jobs.
2. systemctl status: This will show the status of your receiver program.
3. VM instance schedule: This will display the schedule settings for your GCP VM instance.