Google Cloud DataFlow real-time service for batch and stream processing

Shan Zhou, Jay Upadhyay,Wendy Jiang

Summary

Google Cloud Dataflow is the new cloud service that is designed to simplify the mechanics of large-scale data processing, it allows people to concentrate on the logical composition of data processing job, rather than the physical orchestration of parallel processing.

Why use Google Dataflow:

1. It automates the management of processing resources and frees people from operational tasks.
2. On demand, no need to buy reserved compute instances.
3. Automated and optimized work partitioning.
4. Auto scaling of worker resources.
5. Good monitoring using UI and command-line and Stackdriver.
6. Integrating with Cloud Storage, Cloud Pub/Sub, Cloud Datastore, Cloud Bigtable and BigQuery and can be extended to interact with other sources and sinks like Kafka and HDFS
7. Unified Programing Model that enable powerful windowing and correctness control for batch and stream based data sources.

The goal of our project is to provide an overview of the Google Cloud Dataflow and to demonstrate how to build and execute a simple pipeline.

What we have done:

1. Create Storage bucket and installed Cloud SDK in Mac and run an example pipeline remotely using Python.
2. Installed Cloud SDK in Windows and run an example pipeline on Cloud Dataflow Service using Java and Apache Maven
3. Run a mobile gaming pipeline to experience processing in batch and windowing and streaming with Real-Time Game Data. Input source are from Cloud data storage for batch and Pub/Sub for streaming. Results are stored locally and Cloud storage and BigQuery tables.
4. Created an own pipeline using Java and applied pipeline transformation and used google console for monitoring and logs for debugging.

Comparison between spark and google cloud dataflow:

We used a mobile gaming scenario as an example to compare dataflow vs spark in detail using three different kinds of pipelines:

- classic batch pipeline
- window batch pipeline
- streaming pipeline

For more details of this part, please look at comparison dataflow vs spark.pdf

Reference: https://cloud.google.com/dataflow/model/programming-model
https://cloud.google.com/dataflow/docs/

YouTube URL of the full presentation video: https://youtu.be/-2sF5Q0TplA
YouTube URL of the 2min preview presentation video: https://youtu.be/l2eHgQAWdio

Several concepts before we start:

Google Cloud Dataflow:

cloud-based data processing service for both batch and real-time data streaming applications. It expands on earlier Google parallel processing projects, including MapReduce, which originated at the company. It overlaps with competitive software frameworks and services such as Amazon Kinesis, Apache Storm, Apache Spark and Facebook Flux. It agnostically handles data of varying sizes and structures using a format called PCollections, which is short for "parallel collections." It also includes a library of parallel transforms, or PTransforms, which allow high-level programming of often-repeated tasks using basic templates

Google Cloud Storage:

Google Cloud Storage allows world-wide storage and retrieval of any amount of data at any time. You can use Google Cloud Storage for a range of scenarios including serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.

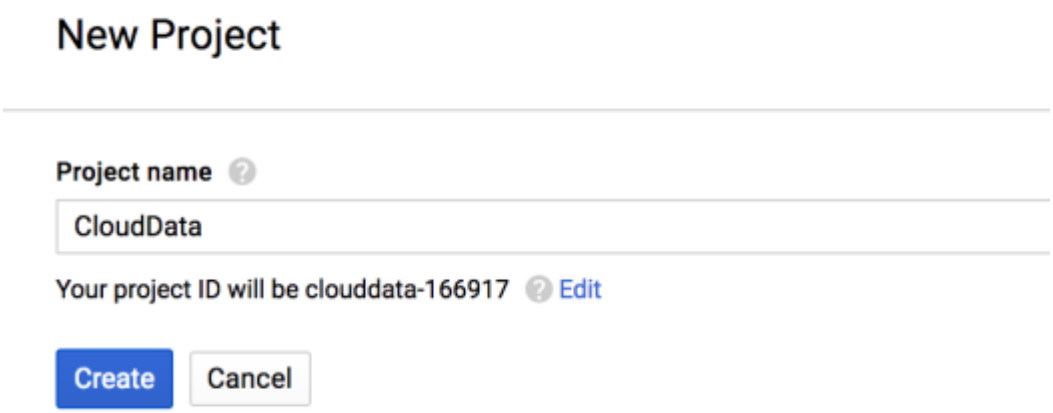
PCollections:
It serves as a persistent and immutable analogue of the Java Collections Framework. This includes efficient, thread-safe, generic, immutable, and persistent stacks, maps, vectors, sets, and bags, compatible with their Java Collections counterparts. Persistent and immutable datatypes are increasingly appreciated as a simple, design-friendly, concurrency-friendly, and sometimes more time- and space-efficient alternative to mutable datatypes.

Windowing:
The Dataflow SDKs use a concept called Windowing to subdivide a PCollection according to the timestamps of its individual elements. Dataflow transforms that aggregate multiple elements, such as GroupByKey and Combine, work implicitly on a per-window basis—that is, they process each PCollection as a succession of multiple, finite windows, though the entire collection itself may be of unlimited or infinite size. The Dataflow SDKs use a related concept called Triggers to determine when to "close" each finite window as unbounded data arrives. Using a trigger can help to refine the windowing strategy for your PCollection to deal with late-arriving data or to provide early results. See Triggers for more information.

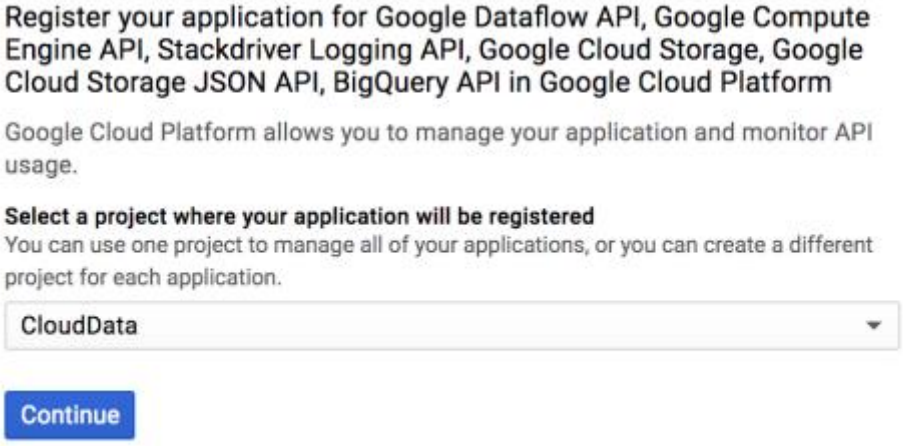
QuickStart using Python in Mac:

1. Create and open an environment for project with name gclouddata and python 2.7:
   ```
   $conda create -n gclouddata python=2.7 anaconda
   $source activate gclouddata
   ```

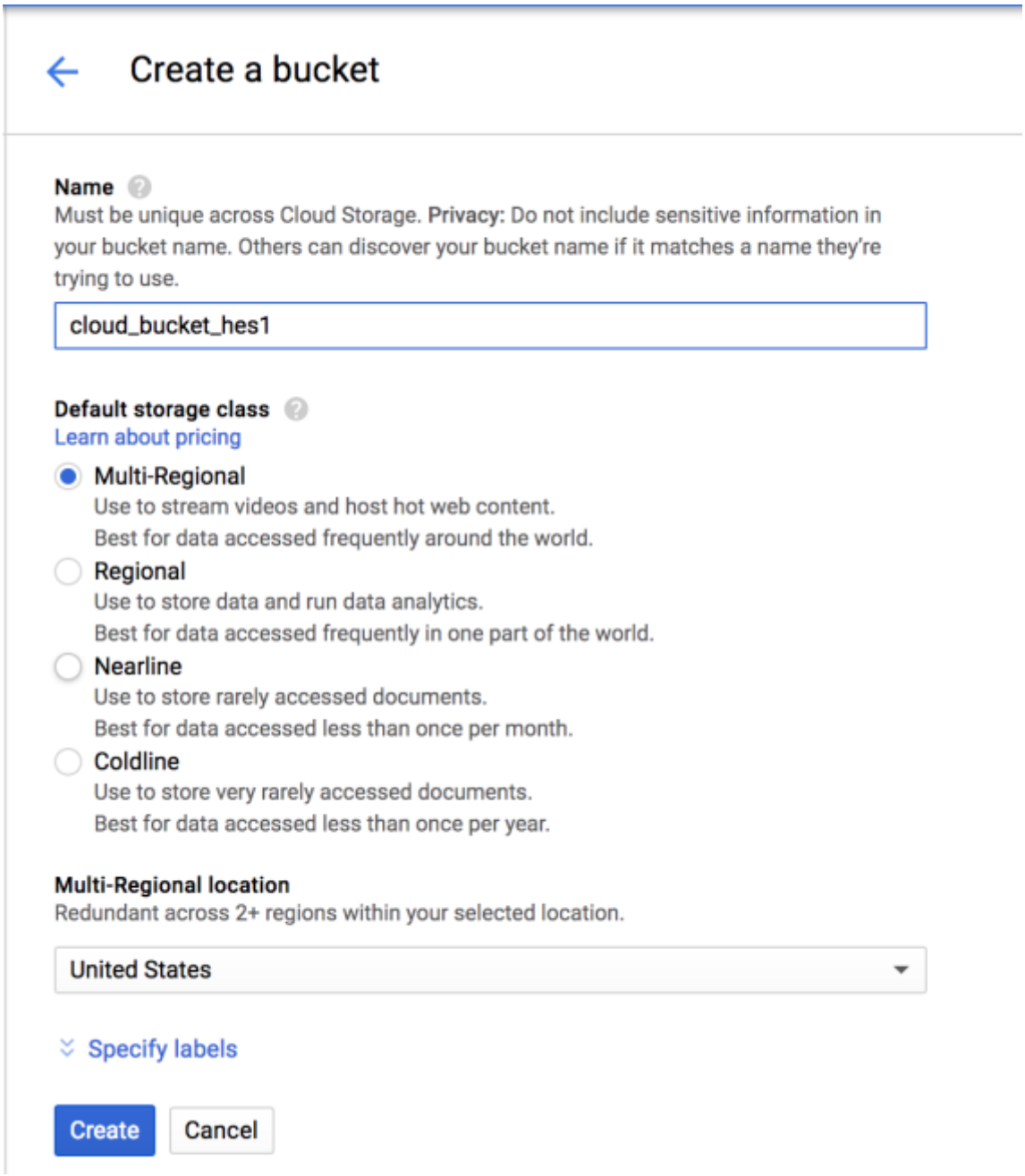
2. Open downloaded folder after extraction google-cloud-sdk:
   Downloaded from https://cloud.google.com/sdk/docs/quickstart-mac-os-x
3. Install using following command:
   ./install.sh
4. Create Project on following link
   https://console.cloud.google.com/cloud-resourcemanager?_ga=1.187232872.1391755138.1492569741



5. Enable APIs:
   https://console.cloud.google.com/flows/enableapi?apiid=dataflow,compute_component,logging,storage_component,storage_api,bigquery&_ga=1.182587750.1391755138.1492569741



6. Create Bucket:
   https://console.cloud.google.com/storage/browser?_ga=1.119224905.1391755138.1492569741



7. Use Command to create new configuration with your email ID , Project name and region:
   gcloud init
8. Install pip :
   pip install -U pip
9. Install google cloud data flow
   pip install google-cloud-dataflow

To Run an Example Pipeline Locally

10. Run Example word-count:
    python -m apache_beam.examples.wordcount --output OUTPUT_FILE

```
(gclouddata) jays-mbp-2:google-cloud-sdk jay$ python -m apache_beam.examples.wor
dcount --output OUTPUT_FILE
No handlers could be found for logger "oauth2client.contrib.multistore_file"
INFO:root:Missing pipeline option (runner). Executing pipeline using the default
 runner: DirectRunner.
INFO:root:Running pipeline with DirectRunner.
INFO:root:Starting finalize_write threads with num_shards: 1, batches: 1, num_th
reads: 1
INFO:root:Renamed 1 shards in 0.13 seconds.
INFO:root:number of empty lines: 1663
```

11. Read Output:

    vi OUTPUT_FILE-00000-of-00001

```
Appear: 1
pardon: 7
justicers: 1
believed: 1
ungovern'd: 1
vermin: 1
needful: 2
foul: 15
Lear: 17
hath: 52
protest: 1
nursery: 1
sleep: 8
hanging: 1
conjuring: 1
garters: 1
appetite: 2
captain: 2
hate: 5
Until: 2
robbers': 1
marching: 1
whose: 15
```

To Run an Example Pipeline Remotely

12. Initialize Variables:

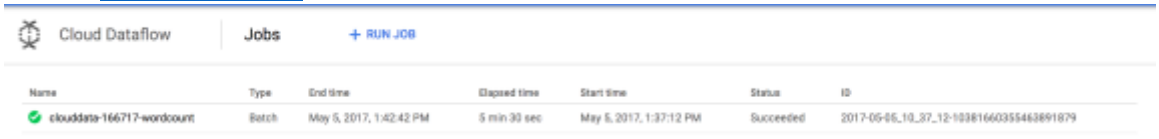
    PROJECT=project name from eg: PROJECT=clouddata-166717

    BUCKET= gs:// <bucket name created in step 6>

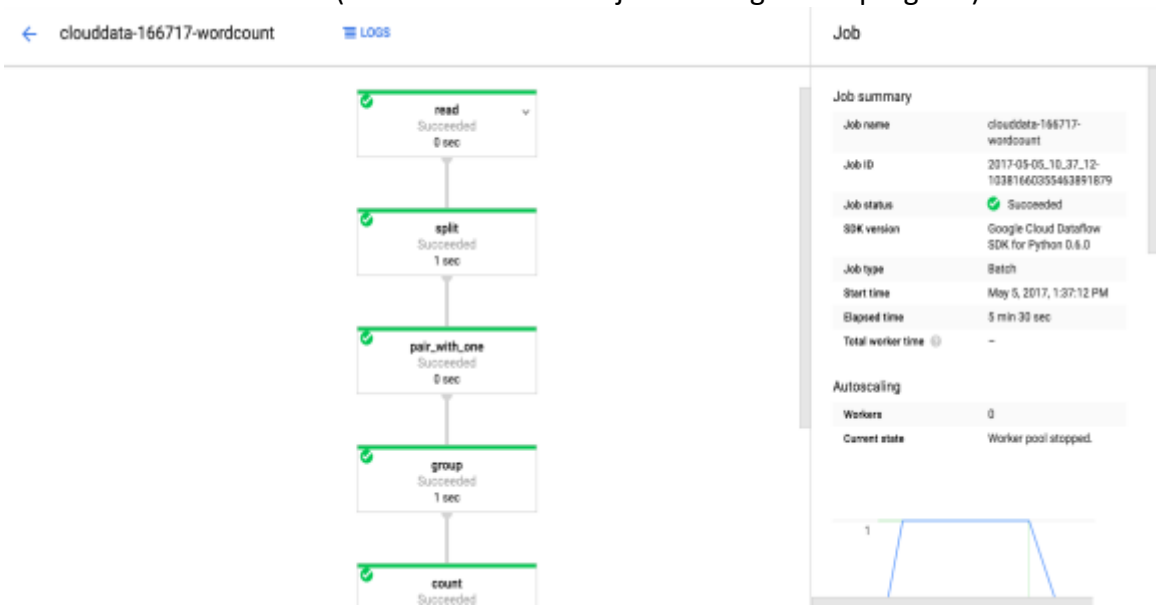
13. Type command:

```
python -m apache_beam.examples.wordcount
--project $PROJECT
--job_name $PROJECT-wordcount
--runner DataflowRunner
--staging_location $BUCKET/staging
--temp_location $BUCKET/temp
 --output $BUCKET/output
```

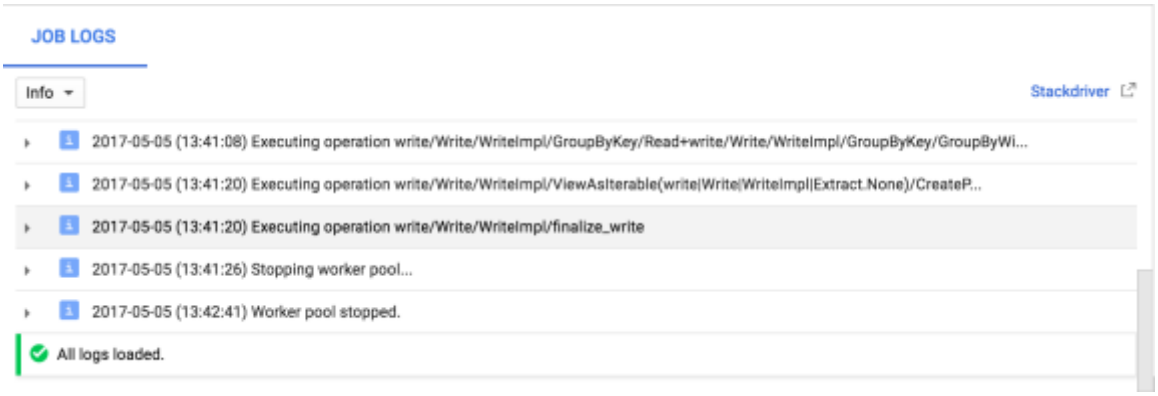
14. After Completion of the run go to:
    https://console.cloud.google.com/dataflow?_ga=1.106519491.1391755138.1492569741



15. Click in the name of Job(can also check while job running to see progress):

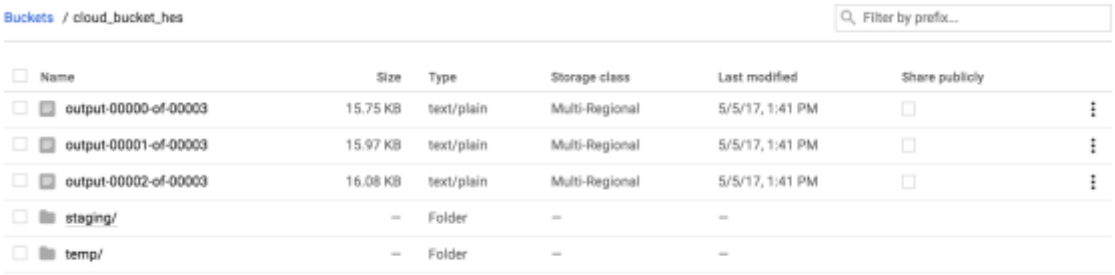


16. Click on logs (can also check while job running to see progress):



17. Go to Cloud storage browser
    https://console.cloud.google.com/storage/browser?_ga=1.144834164.1391755138.1492569741

    In your bucket, you should see the output files and staging files that your job created:



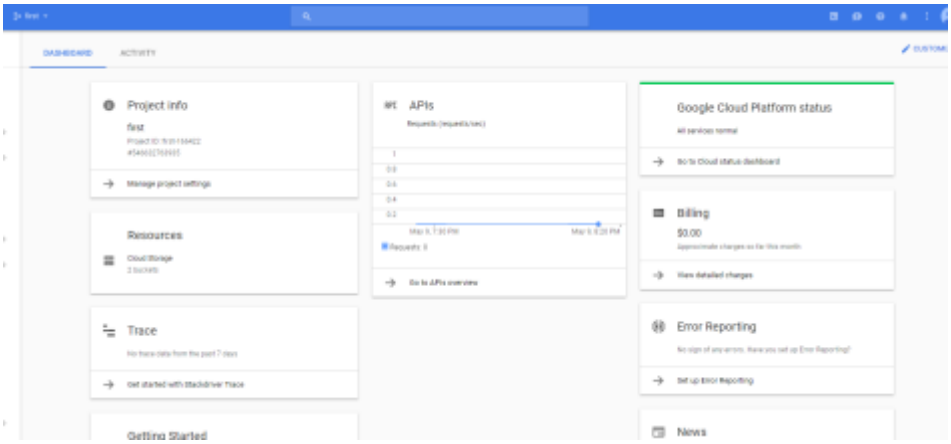
18. Open any output file to see results

```
she: 44
silly: 1
More: 6
believe: 3
Blanket: 1
unfortunate: 1
blot: 1
trunk: 2
Do: 23
remediate: 1
Mean: 1
holla: 1
opposites: 1
holding: 1
discord: 1
cuckoo: 2
entire: 1
dread: 4
riotous: 4
red: 1
changed: 5
Methought: 1
Approach: 1
laid: 1
determine: 2
said: 7
Kneeling: 3
advantage: 1
That's: 7
knowest: 2
enjoy: 2
continent: 1
difference: 4
hideous: 2
o'erwhelm: 1
weeping: 1
May: 9
deeply: 1
step: 2
Tigers: 1
sides: 3
seal: 1
noiseless: 1
Although: 2
kissing: 1
cub: 1
```

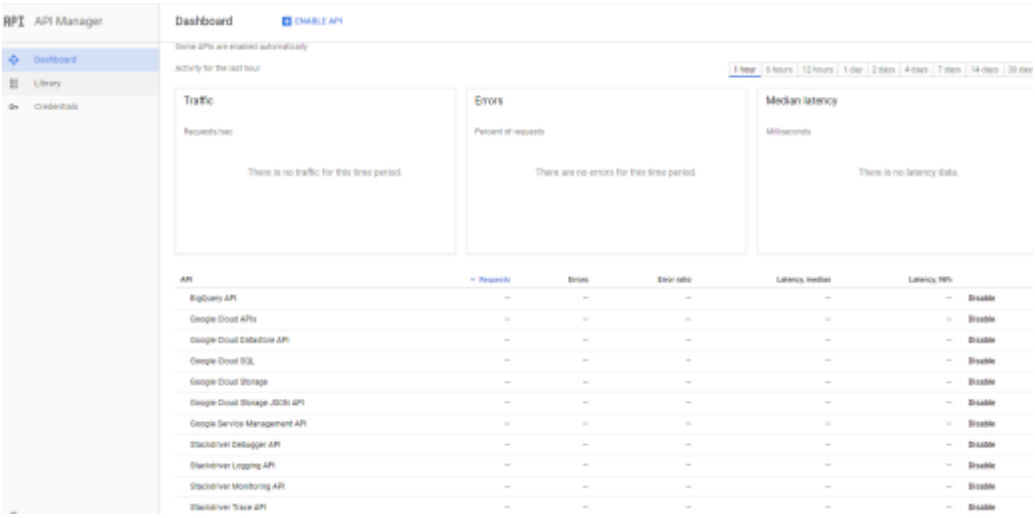
19. Delete the bucket to avoid charges

Quick start using Java Maven on Windows machine:

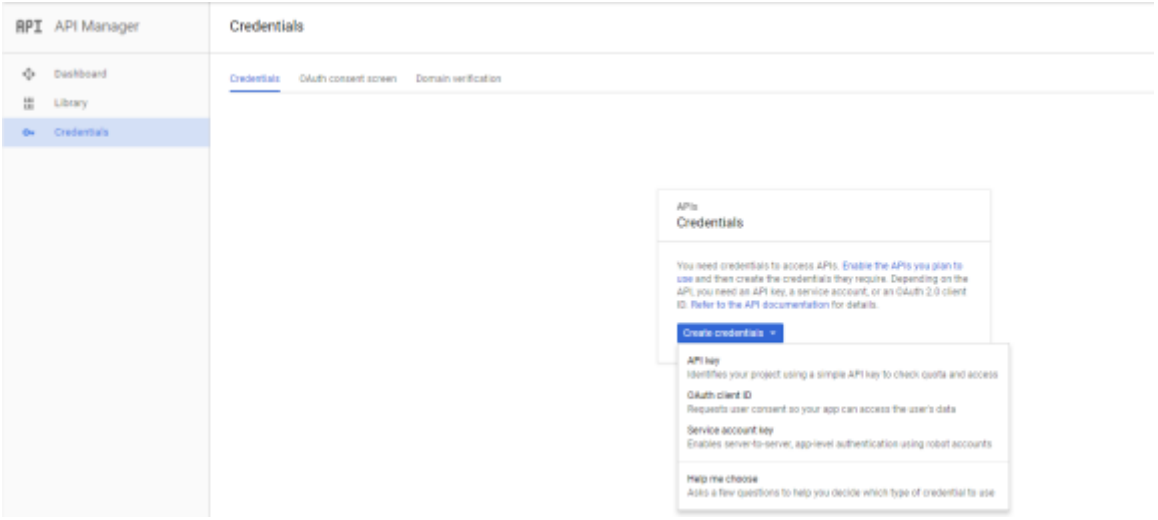
1. In the Google Cloud console, create a project, name ‘first’:

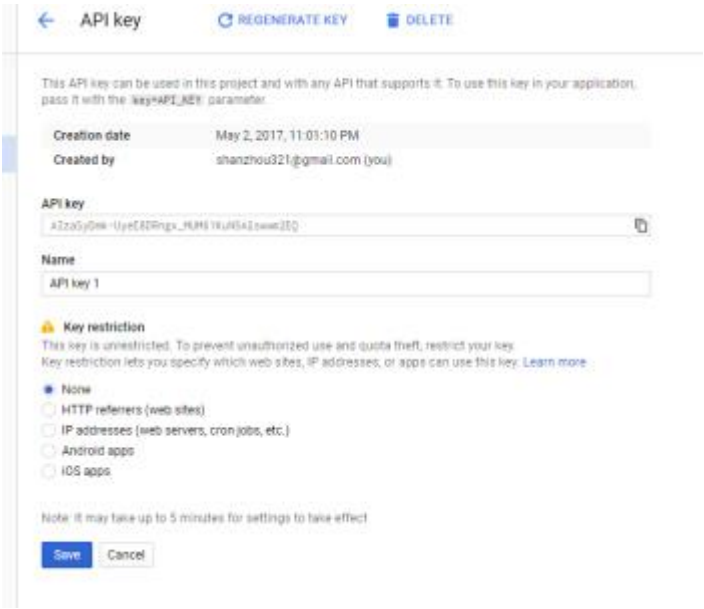


2. In the API management ,make sure Dataflow API is enabled.



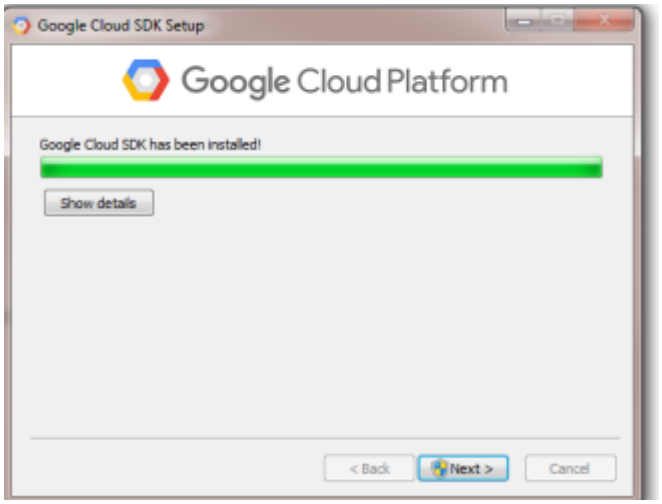
3. Create credentials for API



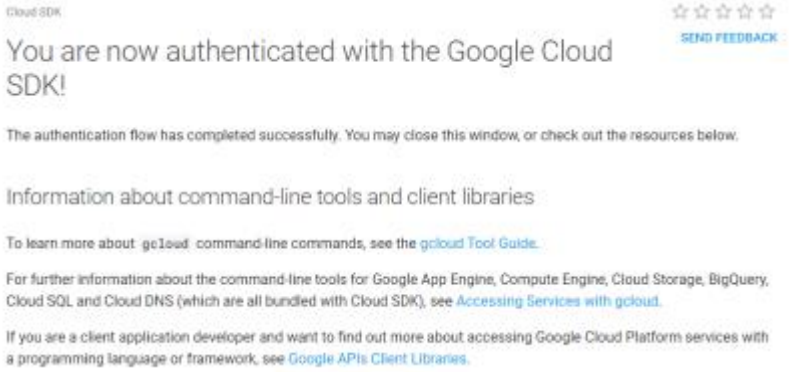


4. Install Google Cloud SDK

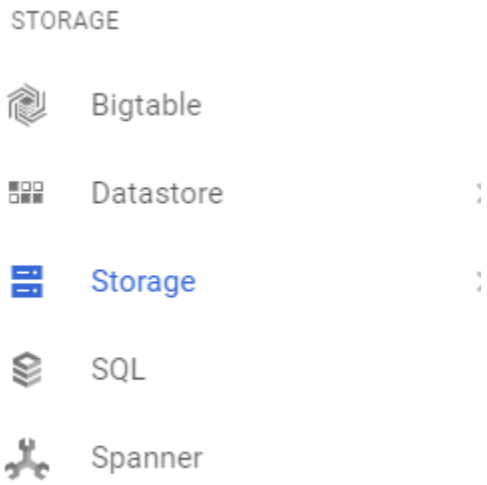
Download the installer from this site: https://cloud.google.com/sdk/docs/ and follow the instruction to install.

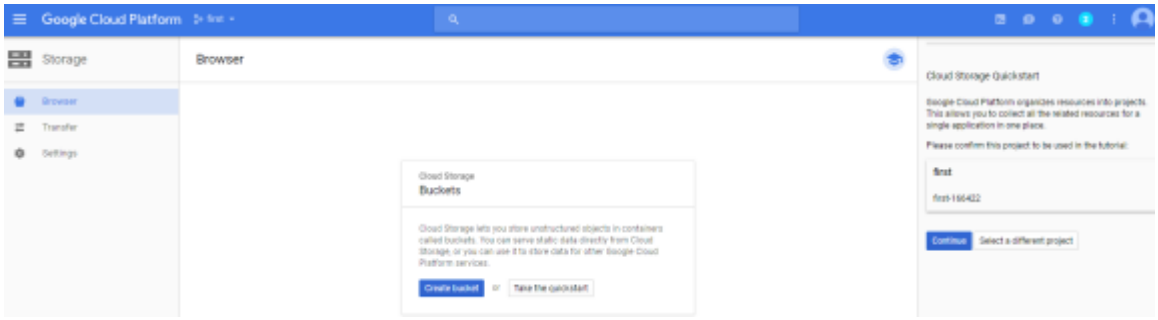


After the installation is done, a web page will be open showing the SDK is good.



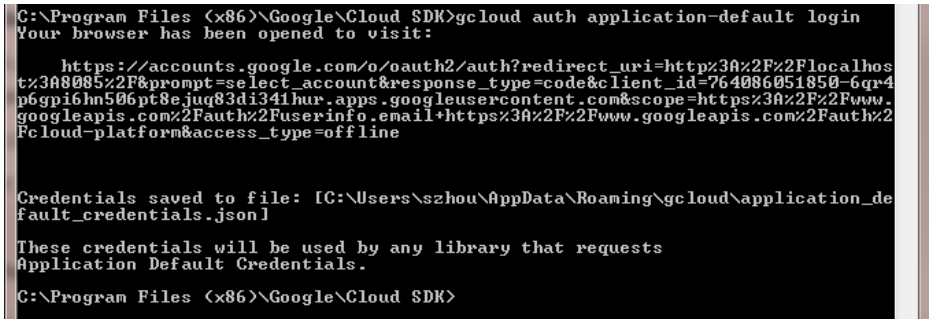
5. In the Storage page, create a bucket name ‘first-166422”. This bucket can be used for both input files, output files and staging files.

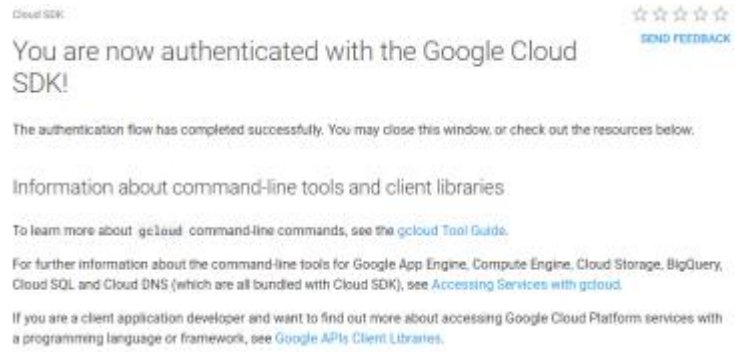




6. Run this command in gcloud command tool to authenticate with the cloud platform.

```
gcloud auth application-default login
```





7. Install Java and set JAVA_HOME environment variable, make sure Java version is 1.7 or 1.8

```
szhou@BOS-1SZHOU-LT C:\Users\szhou
$ java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b15, mixed mode)

szhou@BOS-1SZHOU-LT C:\Users\szhou
$ echo %JAVA_HOME%
C:\java\jdk1.7.0_79

szhou@BOS-1SZHOU-LT C:\Users\szhou
$
```

8. Install Maven and add the bin directory to the path

```
$ mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T15:39:06-04:00)
Maven home: C:\apache-maven-3.5.0\bin\..
Java version: 1.7.0_79, vendor: Oracle Corporation
Java home: C:\java\jdk1.7.0_79\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

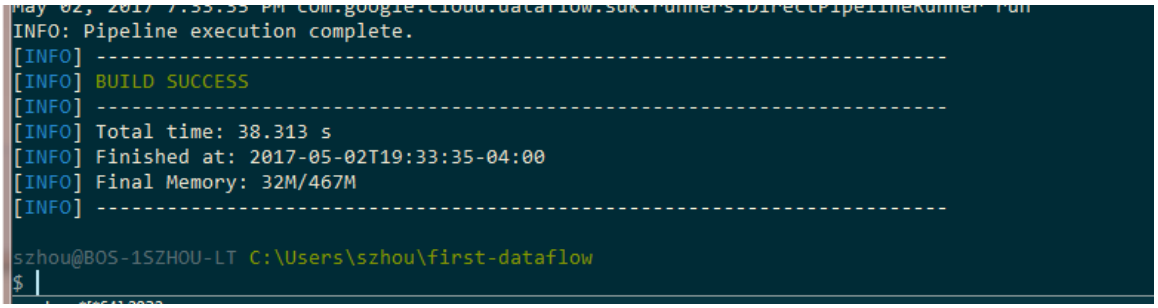
9. Run this following command to create a maven project containing cloud dataflow SDK

```
mvn archetype:generate -DarchetypeArtifactId=google-cloud-dataflow-java-archetypes-examples -DarchetypeGroupId=com.google.cloud.dataflow -DarchetypeVersion=1.9.0 -DgroupId=com.example -DartifactId=first-dataflow -Dversion="0.1" -DinteractiveMode=false -Dpackage=com.example
```

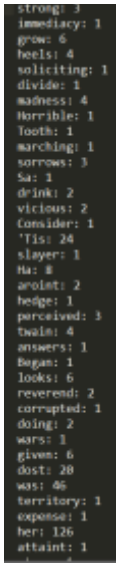
10. Change directory to the project directory just created by Maven, in which there is a pom.xml file. Then build and run example wordcount pipeline using maven. The output data is created and stored locally in a folder named output.

cd first-dataflow

mvn compile exec:java -Dexec.mainClass=com.example.WordCount -Dexec.args="--output=./output/"



11. Open the file in the output folder, here is the word count results:

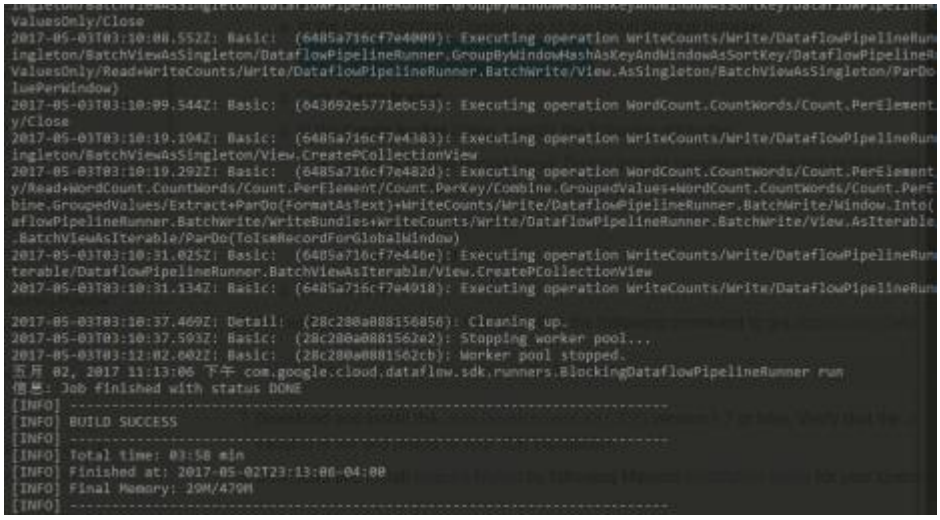


12. Run the word-count on cloud dataflow service. In the arguments, add the google storage bucket and with a staging folder: -Dexec.args="--project=first-166422 . To store the output file in the google storage, add the storage bucket name: -output=gs://first-

166422/output.

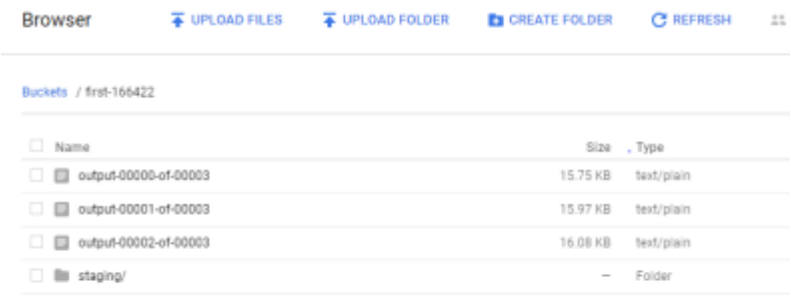
As my project id and bucket name are all first-166422, So I run the following commands:

mvn compile exec:java -Dexec.mainClass=com.example.WordCount -Dexec.args="--project=first-166422 --stagingLocation=gs://first-166422/staging/ --output=gs://first-166422/output --runner=BlockingDataflowPipelineRunner"



13. There is UI monitor in Google platform console to see the status and the step performance and memory and other information .

14. Go to the google storage and open the bucket created, there is staging folder and output files created.



Mobile Game example for batch process, batch process with Windowing and Streaming Process

1. In google console, create a new project name first game, then go to the API manager to enable the DataFlow API. In google storage, create a new bucket for first-game project.

2. Go to BigQuery, create a dataset, which is like a schema in relational database. The output data will be stored as tables in this BigQuery.

3. Login using credential again like quick start instruction.

4. The input for this first-game project is csv files in google cloud platform:

gs://dataflow-samples/game/gaming_data*.csv

The columns are user, team, score, timestamp.

Batch process

5. The UserScore pipeline will sum the score for each user, regardless the timestamp, then write the results into a user_score table in the BigQuery.

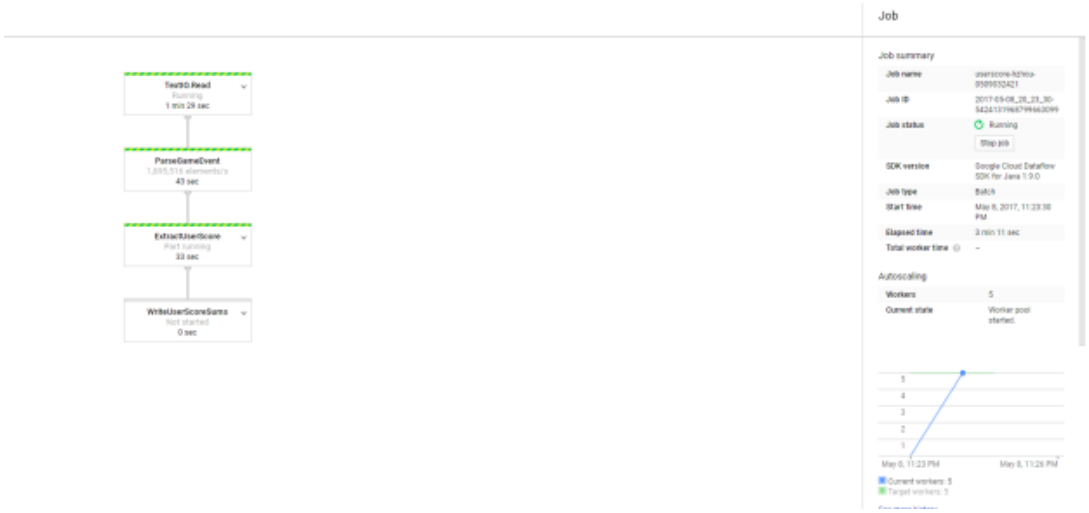
Example code: https://github.com/GoogleCloudPlatform/DataflowJavaSDK-examples/blob/master/src/main/java8/com/google/cloud/dataflow/examples/complete/game/UserScore.java

6. In the maven command line, add arguments: project id, staging location in google storage, dataset in BigQuery, and BlockingDataflowPipelineRunner which will enable the pipeline executed in the cloud using Google Cloud Dataflow service.

mvn compile exec:java -
Dexec.mainClass=com.google.cloud.dataflow.examples.complete.game.UserScore -
Dexec.args="--project=first-game-167101 --stagingLocation=gs://first-game2/staging/ --
dataset=first_dataset --runner=BlockingDataflowPipelineRunner"

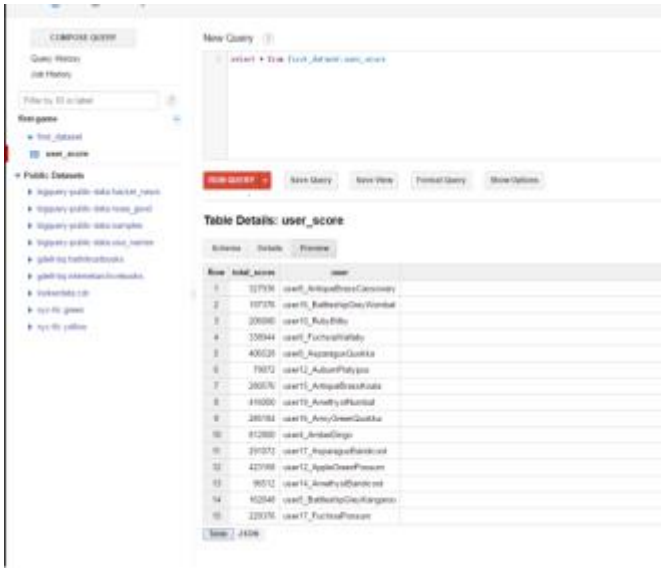
7. After above command runs, in terminal logs, there would be url which will direct to the monitoring in the UI.

8. In the ui Mornitor, we can see the reading file takes 1 minute 29 seconds, parse the file takes 43 seconds, and extract transformation is still running. The workers are 5. It is a very good resource management.



9. After the job complete, check the BigQuery:

A user_score table is created. If I query this table, it will be charged. But we can preview the table, it shows the total score for each user.



Batch Process with windowing

10. The userscore has high latency cause it needs all gaming data has been collected. Then in the HourlyTeamScore, it divides the input data into logical windows and sum the scores for team for each hour. So this pipeline checks the timestamp and ensures it falls within the window. Here is the code that pipeline use the timestamp and window transforms to perform.

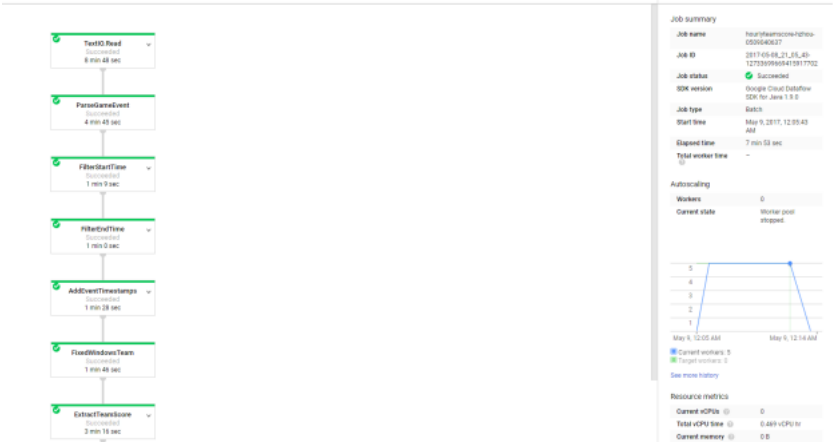
```
.apply("AddEventTimest
amps",
                    WithTimestamps.of((GameActionInfo i) -> new
Instant(i.getTimestamp())))
    .apply(Window.named("FixedWindowsTeam")
        .<GameActionInfo>into(FixedWindows.of(
Duration.standardMinutes(options.getWindowDuration())))
.apply("FilterStartTime", Filter.byPredicate(
   (GameActionInfo gInfo)
      -> gInfo.getTimestamp() >
startMinTimestamp.getMillis()))
```

```
.apply("FilterEndTime", Filter.byPredicate(

    (GameActionInfo gInfo)

        -> gInfo.getTimestamp() <
stopMinTimestamp.getMillis()))
```

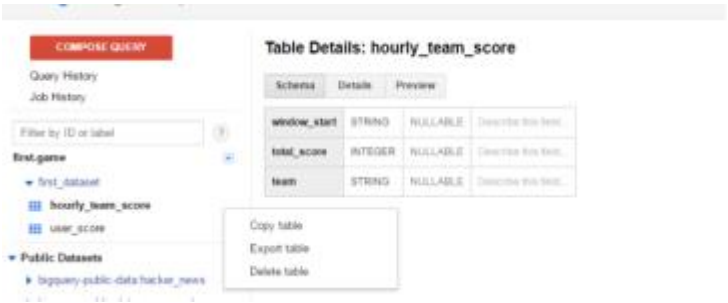
11. Run this command for get hourly scores.

mvn compile exec:java -Dexec.mainClass=com.google.cloud.dataflow.examples.complete.game.HourlyTeamScore -Dexec.args="--project=first-game-167101 --stagingLocation=gs://first-game2/staging/ --dataset=first_dataset --runner=BlockingDataflowPipelineRunner"

12. From UI monitoring, we can see the filtering by start time and end time and fixed window takes 3 minutes:



13. In BigQuery, the hourly_team_score table is created. In the table, for each hour, each team has a total score.



Schema | Details | Preview

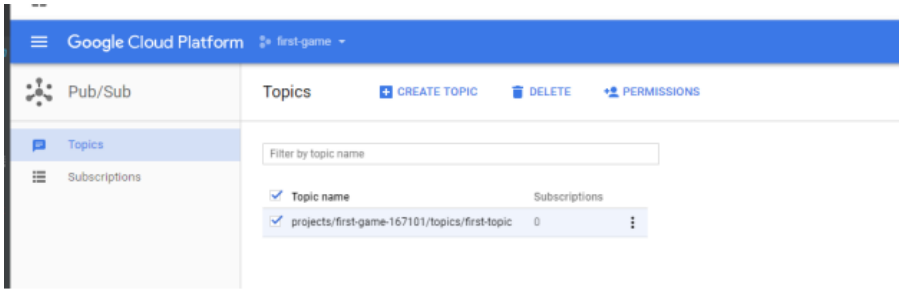
Row	window_start	total_score	team
31	2015-11-16 16:00:00.000	61130	AliceBlueBandicoot
32	2015-11-16 16:00:00.000	1693170	ArmyGreenEmu
33	2015-11-16 17:00:00.000	1838504	AquaQuokka
34	2015-11-16 17:00:00.000	1707180	AliceBlueBandicoot
35	2015-11-16 17:00:00.000	2447112	AlmondPossum
36	2015-11-16 17:00:00.000	1967870	AmaranthBilby
37	2015-11-16 17:00:00.000	2642552	AmaranthMarmot
38	2015-11-16 17:00:00.000	2307501	FuchsiaWombat
39	2015-11-16 17:00:00.000	1160180	AndroidGreenPlatypus
40	2015-11-16 17:00:00.000	1284061	AmethystMarmot
41	2015-11-16 17:00:00.000	473757	BisqueEmu
42	2015-11-16 17:00:00.000	1079844	AmethystBilby
43	2015-11-16 17:00:00.000	883782	BananaDingo
44	2015-11-16 17:00:00.000	606670	AmethystCassowary
45	2015-11-16 17:00:00.000	482718	AntiqueBrassBandicoot

Table | JSON

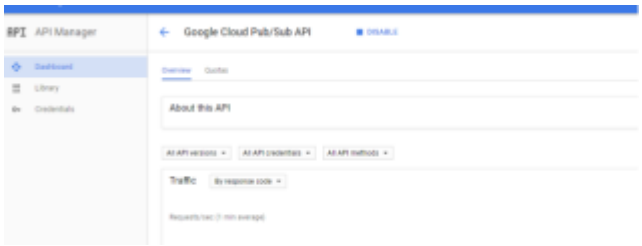
Streaming Process

14. The streaming pipeline use Google Cloud Pub/Sub as an input source, the game server publish the score data to Pub/Sub. Google Cloud Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications.

First, we need to create a topic in Pub/Sub



Then enable the Pub/Sub API in API Manager:



15. The streaming pipeline calculated cumulative user score every ten minutes. It uses an

unbounded data source, which let us to handle late data with lower latency.

Here is the code that calculates user score based on processing time:

```
 * Extract user/score pairs from the event stream using processing time, via global windowing.
 * Get periodic updates on all users' running scores.
 */
@VisibleForTesting
static class CalculateUserScores
    extends PTransform<PCollection<GameActionInfo>, PCollection<KV<String, Integer>>> {
  private final Duration allowedLateness;

  CalculateUserScores(Duration allowedLateness) {
    this.allowedLateness = allowedLateness;
  }

  @Override
  public PCollection<KV<String, Integer>> apply(PCollection<GameActionInfo> input) {
    return input.apply("LeaderboardUserGlobalWindow",
        Window.<GameActionInfo>into(new GlobalWindows())
          // Get periodic results every ten minutes.
          .triggering(Repeatedly.forever(AfterProcessingTime.pastFirstElementInPane()
              .plusDelayOf(TEN_MINUTES)))
          .accumulatingFiredPanes()
          .withAllowedLateness(allowedLateness))
        // Extract and sum username/score pairs from the event data.
        .apply("ExtractUserScore", new ExtractAndSumScore("user"));
```

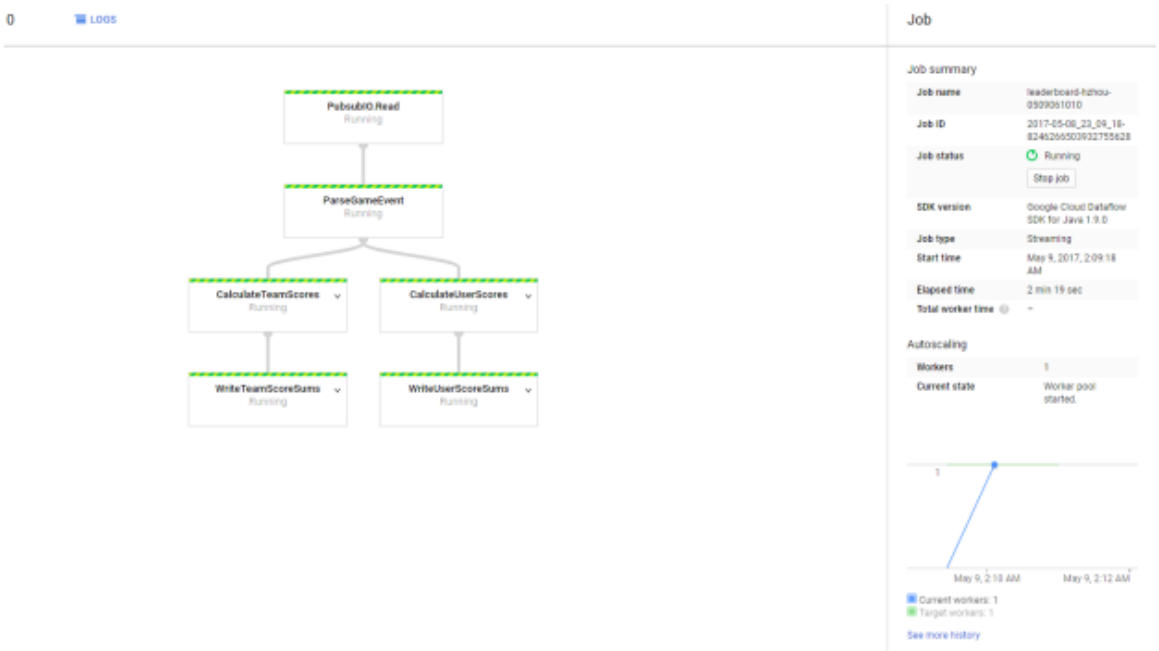
}

}

16. To run the code, we need to add the topic for Pub/Sub as a parameter.

mvn compile exec:java -
Dexec.mainClass=com.google.cloud.dataflow.examples.complete.game.LeaderBoard -
Dexec.args="--project=first-game-167101 --stagingLocation=gs://first-game2/staging/ --
dataset=first_dataset --topic=projects/first-game-167101/topics/first-topic --
runner=BlockingDataflowPipelineRunner"

17. The job keeps running for a whole day until I manually stop the job.



Create Own Pipeline to get top score per user

1. Create the a pipeline options object, which is configured by command line flags.

Options options = PipelineOptionsFactory.fromArgs(args).withValidation().as(Options.class);

Pipeline pipeline = Pipeline.create(options);

2. Create interface options to get parameter for input, Pub/Sub dataset, and output table name for BigQuery table:

```
public interface Options extends PipelineOptions {


  @Description("Path to the data file(s) containing game data.")
  @Default.String("gs://dataflow-samples/game/gaming_data*.csv")
  String getInput();
  void setInput(String value);


  @Description("BigQuery Dataset to write tables to. Must already exist.")
  @Validation.Required
  String getDataset();
  void setDataset(String value);


  @Description("The BigQuery table name. Should not already exist.")
  @Default.String("user_top2_scores")
  String getTableName();
  void setTableName(String value);
}
```

3. Build transformation PCollection to get the top 2 score for each user and convert the list of top 2 score into string.

```
/**
 * A transform to extract get the top 2 scores for each 'team' or 'user. The
 * constructor arg determines whether 'team' or 'user' info is extracted.
 */
// [START DocInclude_USExtractXform]
public static class ExtractMaxScore
    extends PTransform<PCollection<GameActionInfo>, PCollection<KV<String, String>>> {
```

```
  private final String field;


  ExtractMaxScore(String field) {
    this.field = field;
  }


  @Override
  public PCollection<KV<String, String>> apply(PCollection<GameActionInfo> gameInfo) {


    return gameInfo.apply(MapElements.via((GameActionInfo gInfo) -> KV.of(gInfo.getKey(field),
gInfo.getScore())).withOutputType(new TypeDescriptor<KV<String, Integer>>() {
    }))
        .apply(Top.largestPerKey(2))
        .apply(MapElements.via((KV<String, List<Integer>> entry) -> KV.of(entry.getKey(),
Joiner.on(",").join(entry.getValue()))).withOutputType(new TypeDescriptor<KV<String,
String>>() {
        }));
  }
 }
 // [END DocInclude_USExtractXform]
```

4. Write the user and top 2 scores into BigQuery table with columns as user and top2:

```
  protected static Map<String, WriteToBigQuery.FieldInfo<KV<String,  String>>>
  configureBigQueryWrite() {
  Map<String, WriteToBigQuery.FieldInfo<KV<String,  String>>> tableConfigure =
      new HashMap<>();
  tableConfigure.put("user",
```

```
      new WriteToBigQuery.FieldInfo<KV<String,  String>>("STRING", c -> c.element().getKey()));
   tableConfigure.put("top2",
      new WriteToBigQuery.FieldInfo<KV<String,  String>>("STRING", c ->
c.element().getValue()));
   return tableConfigure;
  }
```

5. In main function, apply these transformation and write data for the pipeline, and provide a name for each step so that we can view the step performance in UI monitoring:

```
   public static void main(String[] args) throws Exception {
   // Begin constructing a pipeline configured by commandline flags.
   Options options = PipelineOptionsFactory.fromArgs(args).withValidation().as(Options.class);
   Pipeline pipeline = Pipeline.create(options);

   // Read events from a text file and parse them.
   pipeline.apply(TextIO.Read.from(options.getInput()))
     .apply(ParDo.named("ParseGameEvent").of(new ParseEventFn()))
     // Extract and sum username/score pairs from the event data.
     .apply("ExtractTeamTopScore", new ExtractMaxScore("user"))
     .apply("WriteTeamRank",
        new WriteToBigQuery<KV<String, String>>(options.getTableName(),
                                    configureBigQueryWrite()));

   // Run the batch pipeline.
   pipeline.run();
  }
```

Instructions to run this code:

1. Follow above Java quick start documentation to install SDK, enable DataFlow API, create project, create bucket, create BigQuery dataset and create credential in google cloud console.

2. Unzip the DataFlowCode.zip

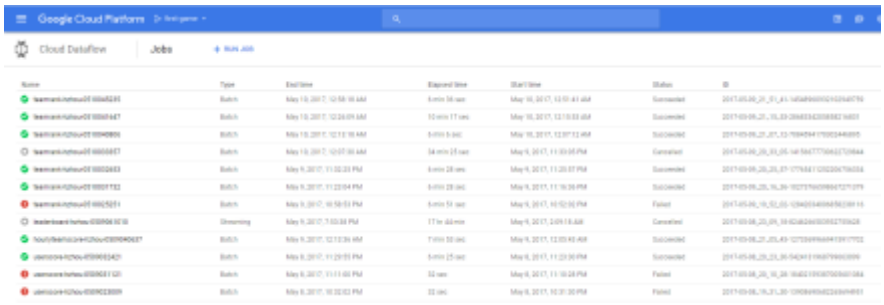
3. In command, change directory to DataFlowCode directory

4. Run the following command, if your project id, google storage name, BigQuery dataset name is different than the following command, please modify the command:

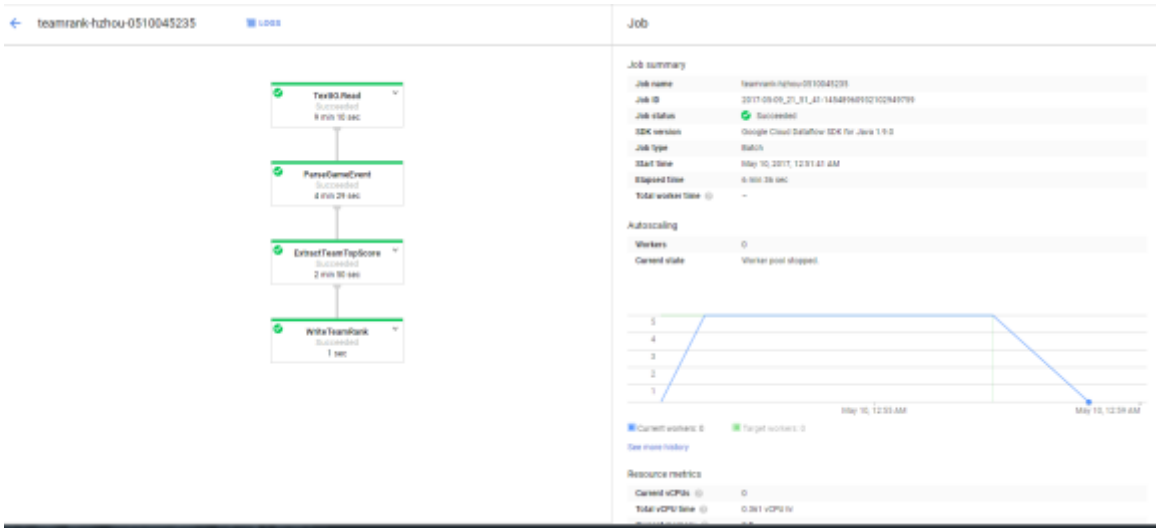
mvn compile exec:java -Dexec.mainClass=cscie63.dataflow.topscore.Top2Score -Dexec.args="--project=first-game-167101 --stagingLocation=gs://first-game2/staging/ --dataset=first_dataset --runner=BlockingDataflowPipelineRunner"

5. You should see this information in the terminal:

6. In the google console, under your project jobs, get click the latest job:



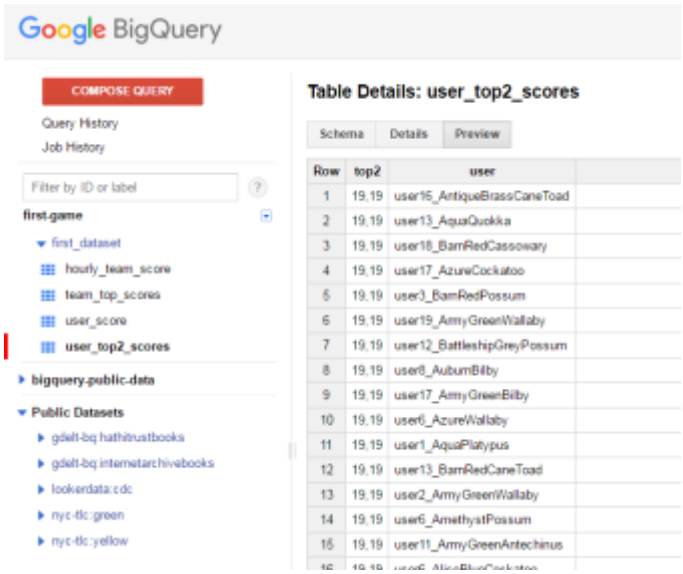
It will show the step performance and status:



7. After a few minutes running, in terminal, you should see build success:

```
2017-05-10T04:56:47.056Z: Detail:  (5dba6e9701439658): Cleaning up.
2017-05-10T04:56:47.063Z: Basic:  (5dba6e9701439df4): Stopping worker pool...
2017-05-10T04:58:17.080Z: Basic:  (5dba6e9701439ba9): Worker pool stopped.
五月 10, 2017 12:59:31 上午 com.google.cloud.dataflow.sdk.runners.BlockingDataflowPipelineRunner run
信息: Job finished with status DONE
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 06:58 min
[INFO] Finished at: 2017-05-10T08:59:31-04:00
[INFO] Final Memory: 35M/590M
[INFO] ------------------------------------------------------------------------

C:\Users\hzhou\Downloads\big data analytics\final project\DataFlowCode
```

8. Then go to BigQuery, you should see the user_top2_scores table has been created, and you can preview data for free.



Comparison: spark vs google cloud dataflow

Dataflow is unique amongst data parallel systems in that it is built upon a comprehensive model for out-of-order processing, it's designed to meet the challenges of real-time data processing without compromising correctness.

We used a mobile gaming scenario as an example to compare dataflow vs spark in detail.

There are four use-cases:

- **User Scores** - A classic batch pipeline calculating per-user scores over a bounded set of input data.
- **Hourly Team Scores** - A batch pipeline calculating per-hour, per-team scores over a bounded set of input data.
- **Leaderboard** - A streaming pipeline continuously calculating two types of scores: per-hour, per-team scores as before, and cumulative per-user score totals over all time.
- **Game Stats** - A streaming pipeline computing spam-filtered, per-hour, per-team scores, as well as a more complex hourly analysis of average per-user engagement time for the game.

We kind of get the expression that dataflow provides the flexibility and power necessary for the next generation of real-time data-processing systems, with a clear, practical, and robust approach to out-of-order processing. It lets us write clean, modular code that evolves beautifully over time as needs change and expand. it looks very promising even through it's still very young. The model maps directly onto the four questions that are relevant in any out-of-order data processing pipeline:

- What results are calculated? Answered via transformations.
- Where in event time are results calculated? Answered via event-time windowing.
- When in processing time are results materialized? Answered via watermarks, triggers, and allowed lateness.
- How do refinements of results relate? Answered via accumulation modes

Conclusion:

This Google Dataflow is a very powerful tool to develop and execute different data processing patterns including ETL, batch computation and continuous computation. It also eliminates programing model switching cost.Besides, It integrates with Cloud Storage, Cloud Pub.Sub, Cloud Datastore, Cloud Bigtable and BigQuery. Furthermore, it has very good monitoring integrated into Google Cloud Platform console.

However, it does not have many example and python modules. Right now it is better for user to use Java instead of Python. But if user is not very familiar with Java, it will take a lot of time to write the pipeline.

Reference:

https://cloud.google.com/dataflow/

https://cloud.google.com/dataflow/docs/

https://cloud.google.com/dataflow/blog/dataflow-beam-and-spark-comparison
http://www.infoworld.com/article/3064728/analytics/google-cloud-dataflow-vs-apache-spark-benc hmarks-are-in.html
http://stackoverflow.com/questions/33518104/google-dataflow-vs-apache-spark

https://cloud.google.com/dataflow/docs/quickstarts/quickstart-python

https://cloud.google.com/sdk/docs/quickstart-mac-os-x