- CoGroupByKey: For a relational join of two or more key/value PCollections with the same key type
- Combine: For combining collections of elements or values in your data
- Flatten: For merging multiple PCollection objects
- Partition: Splits a single PCollection into smaller collections

- I/O transforms: These are PTransforms that read or write data to different external storage systems. Some of the currently available I/O transforms working with Beam Python SDK include
    - avroio: For reading from and writing to an Avro file
    - textio: For reading from and writing to text files

For a simple linear pipeline with sequential transformation, the processing graph looks like what is shown in Figure 40-1.

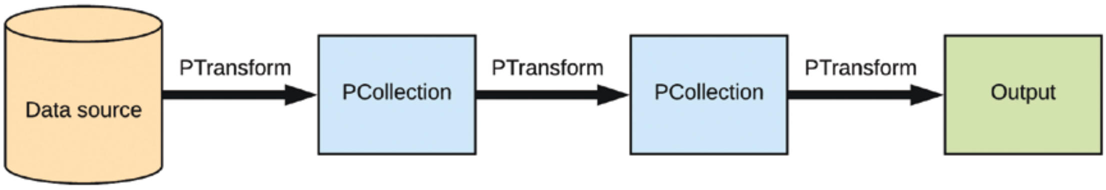


Figure 40-1. *A simple linear Pipeline with sequential transforms*

Building a Simple Data Processing Pipeline

In this simple Beam application, we will build a Dataflow pipeline to preprocess a CSV file from a GCS bucket and write the output back to GCS. This example selects certain features and rows that are of interest to the downstream modeling task. Here, we considered the ‘crypto-markets.csv’ dataset. In the data preprocessing pipeline, we removed data attributes that may not be relevant for analytics/model building and we also filtered records pertaining to 'bitcoin'. The steps that follow create a simple Beam pipeline and execute in on Google Dataflow:

1. Enable the GCP Cloud Dataflow API and Cloud Resource Manager API from the APIs & Services dashboard.
2. Open a new Notebook.
3. Note that at this time of writing, Apache Beam only works with Python version 2.7, so be sure to switch the kernel for your Python interpreter. Add the following code blocks in the Notebook cell.
4. If running on Google Colab, first authenticate the notebook with GCP.

   ```
   from google.colab import auth
   auth.authenticate_user()
   print('Authenticated')

   # configure GCP project. Change to your project ID
   project_id = 'ekabasandbox'
   !gcloud config set project {project_id}
   ```

5. Install the Apache beam library and other important setup packages.

   ```
   %%bash
   pip install apache-beam[gcp]
   ```

6. After installing, change the notebook runtime type to Python 2.
7. Next, reset the notebook kernel before running the code to import the relevant libraries.

   ```
   import apache_beam as beam
   from apache_beam.io import ReadFromText
   from apache_beam.io import WriteToText
   ```

8. Assign the parameters for the pipeline. Replace the relevant parameters with your entries.

   ```
   # parameters
   staging_location = 'gs://enter_bucket_name/staging' # change this
   temp_location = 'gs://enter_bucket_name/temp' # change this
   ```

```
job_name = 'dataflow-crypto'
project_id = enter_project_id' # change this
source_bucket = 'enter_bucket_name' # change this
target_bucket = 'enter_bucket_name' # change this
```

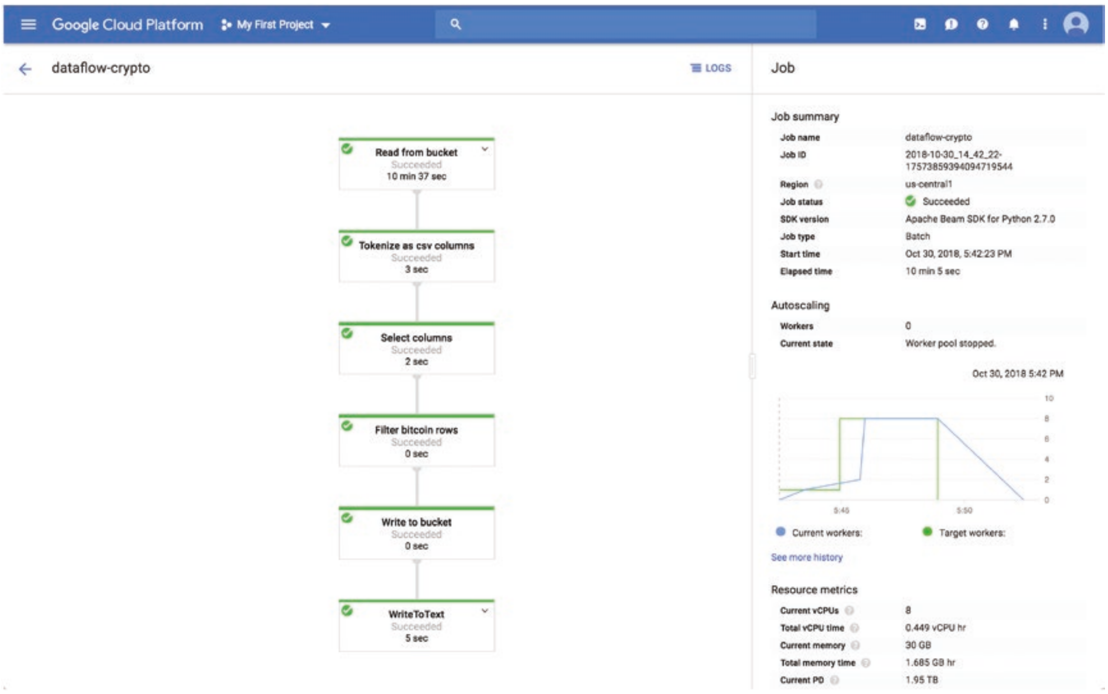
9. Method to build and run the pipeline.

```
def run(project, source_bucket, target_bucket):
    import csv

    options = {
        'staging_location': staging_location,
        'temp_location': temp_location,
        'job_name': job_name,
        'project': project,
        'max_num_workers': 24,
        'teardown_policy': 'TEARDOWN_ALWAYS',
        'no_save_main_session': True,
        'runner': 'DataflowRunner'
      }
    options = beam.pipeline.PipelineOptions(flags=[], **options)

    crypto_dataset = 'gs://{}/crypto-markets.csv'.format(source_
    bucket)
    processed_ds = 'gs://{}/transformed-crypto-bitcoin'.
    format(target_bucket)

    pipeline = beam.Pipeline(options=options)

    # 0:slug, 3:date, 5:open, 6:high, 7:low, 8:close
    rows = (
        pipeline |
            'Read from bucket' >> ReadFromText(crypto_dataset) |
            'Tokenize as csv columns' >> beam.Map(lambda line:
            next(csv.reader([line]))) |
            'Select columns' >> beam.Map(lambda fields:
            (fields[0], fields[3], fields[5], fields[6],
            fields[7], fields[8])) |
                'Filter bitcoin rows' >> beam.Filter(lambda row: row[0] ==
                'bitcoin')
            )

        combined = (
            rows |
                'Write to bucket' >> beam.Map(lambda (slug, date,
                open, high, low, close): '{},{},{},{},{},{}'.format(
                    slug, date, open, high, low, close)) |
                WriteToText(
                    file_path_prefix=processed_ds,
                    file_name_suffix=".csv", num_shards=2,
                    shard_name_template="-SS-of-NN",
                    header='slug, date, open, high, low, close')
            )

        pipeline.run()
```

10. Run the pipeline.

```
if __name__ == '__main__':
     print 'Run pipeline on the cloud'
     run(project=project_id, source_bucket=source_bucket,
     target_bucket=target_bucket)
```

The image in Figure 40-2 shows the Dataflow pipeline created as a result of this job.



Job

Job summary

Job name	dataflow-crypto
Job ID	2018-10-30_14_42_22-17573859394094719544
Region	us-central1
Job status	✓ Succeeded
SDK version	Apache Beam SDK for Python 2.7.0
Job type	Batch
Start time	Oct 30, 2018, 5:42:23 PM
Elapsed time	10 min 5 sec

Autoscaling

Workers	0
Current state	Worker pool stopped.

Resource metrics

Current vCPUs	8
Total vCPU time	0.449 vCPU hr
Current memory	30 GB
Total memory time	1.685 GB hr
Current PD	1.95 TB

Figure 40-2. *Preprocessing Pipeline on Google Cloud Dataflow*

More complex and advanced uses of Google Cloud Dataflow are beyond the scope of this book as they are more in the area of building big data pipelines for large-scale data transformation. However, this section is included because big data transformation is an important component for the design and productionalization of machine learning models when solving a particular business use case at scale. It is important for readers to get a feel of working with these sort of technologies.

This chapter provides an introduction to building large-scale big data transformation pipelines using Python Apache Beam programming model that runs on Google Dataflow computing infrastructure. The next chapter will cover using Google Cloud Machine Learning Engine to train and deploy large-scale models.

CHAPTER 41

Google Cloud Machine Learning Engine (Cloud MLE)

The Google Cloud Machine Learning Engine, simply known as Cloud MLE, is a managed Google infrastructure for training and serving "large-scale" machine learning models. Cloud ML Engine is a part of GCP AI Platform. This managed infrastructure can train large-scale machine learning models built with TensorFlow, Keras, Scikit-learn, or XGBoost. It also provides modes of serving or consuming the trained models either as an online or batch prediction service. Using online prediction, the infrastructure scales in response to request throughout, while with the batch mode, Cloud MLE can provide inference for TBs of data.

Two important features of Cloud MLE is the ability to perform distribution training and automatic hyper-parameter tuning of your models while training. The big advantage of automatic hyper-parameter tuning is the ability to find the best set of parameters that minimize the model cost or loss function. This saves time of development hours in iterative experiments.

The Cloud MLE Train/Deploy Process

The high-level overview of the train/deploy process on Cloud MLE is depicted in Figure 41-1:

1. The data for training/inference is kept on GCS.
2. The execution script uses the application logic to train the model on Cloud MLE using the training data.

© Ekaba Bisong 2019
E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_41