Learn how to implement large-scale data pipelines quickly and easily using the Google Cloud. We'll demonstrate how to build pipelines that collect source data in the App Engine Datastore and Google Cloud Storage, process and transform it using MapReduce, and run ad-hoc analysis with Google BigQuery

About Michael Manoochehri

View full profile

Michael is a Developer Programs Engineer supporting developers who work with Google's Cloud platform. With many years of experience working for research and non-profit organizations, he is interested in making Big Data analysis more accessible and affordable.

Building Data Pipelines at Google Scale

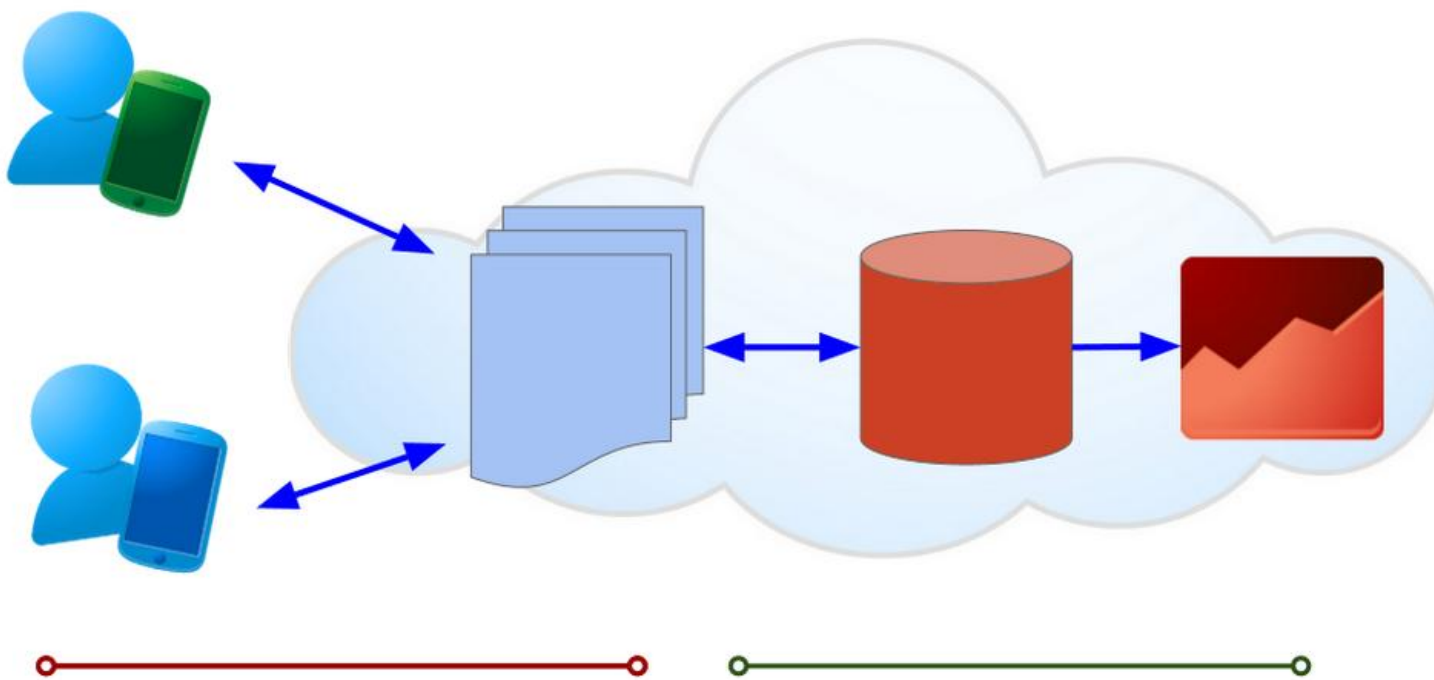
Michael Manoochehri
Developer Programs Engineer, Google

Google I/O 12

In this talk...

- What is a Data Pipeline?
- Look at technologies involved:
  - App Engine Pipeline API, MapReduce Library
  - App Engine Datastore
  - Google BigQuery
  - Google Cloud Storage
- Use Cases: Building Data Pipelines using App Engine
- Best Practices
- Q & A

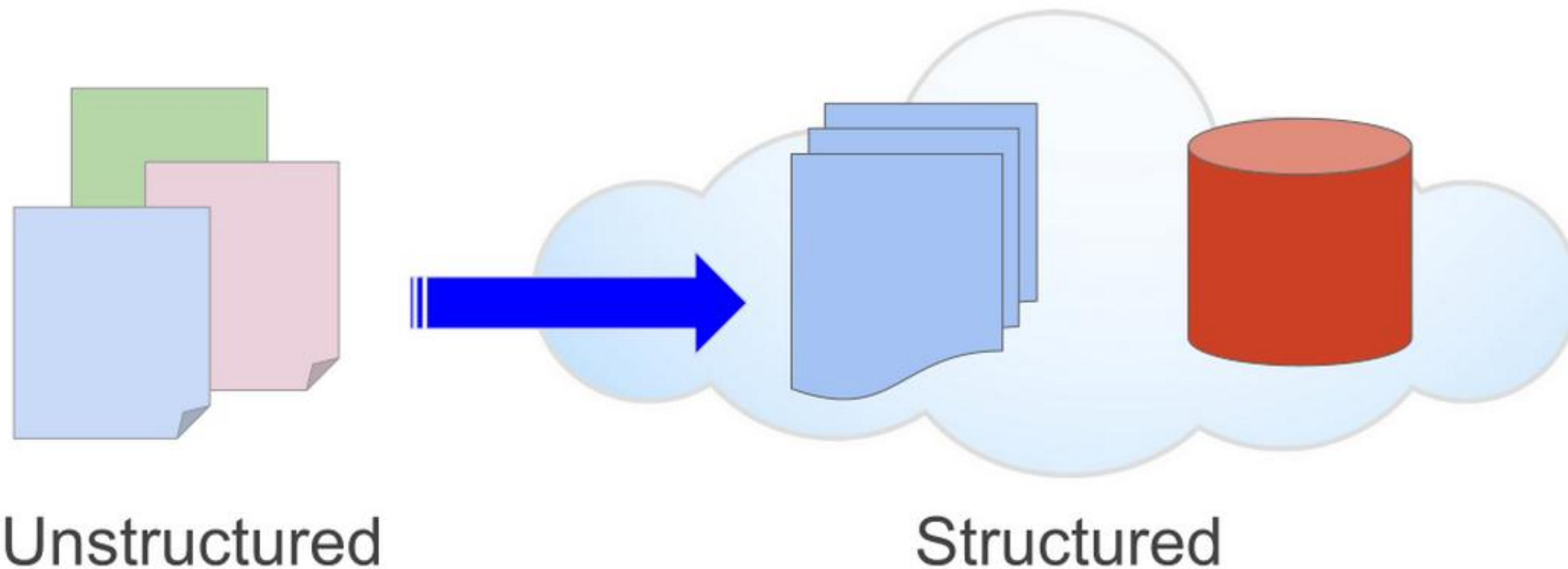
Data Application Patterns



High avaliability and performance

Store and ask questions about massive amounts of data

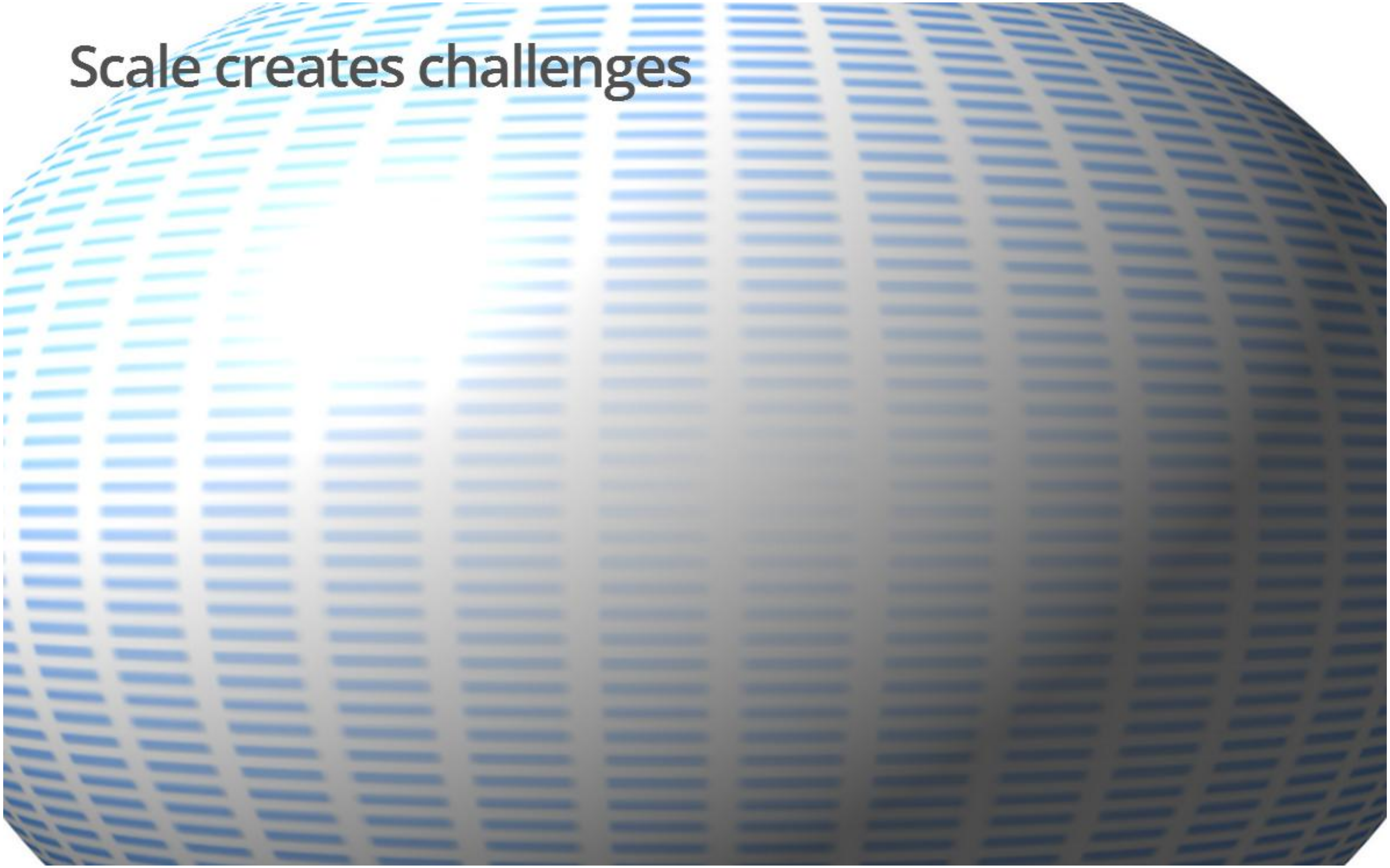
Data Application Patterns



Your Data

```
= {'name':'Michael', 'employer': 'Google'}
```

Scale creates challenges



Massive Datasets: Right Technology for the Task

NoSQL database	High availability and performance
Analysis tools	Real-time aggregate queries
Ubiquitous Storage	Archiving, scale, data transformations

Google's Cloud Data Services

App Engine Datastore

Collect and serve data at scale

- A `{'key':'value'}` Store (with indexes)
- Highly available and high performance
- Allows for CRUD operations on records
- Fluid Schema

“SQL vs NoSQL: Battle of the Backends”

Alfred Fuller, Ken Ashcraft

Google BigQuery

- SQL-like queries on massive datasets
- via a RESTful API
- Append-only
- Fixed Schema (unlike the fluid schema of Datastore)
- Data ingestion via CSV data
- Demo: BigQuery Web UI
- Demo: QlikView US Natality Data Dashboard

“ Crunching Big Data with BigQuery ”

Ryan Boyd, Jordan Tigani

Google Cloud Storage

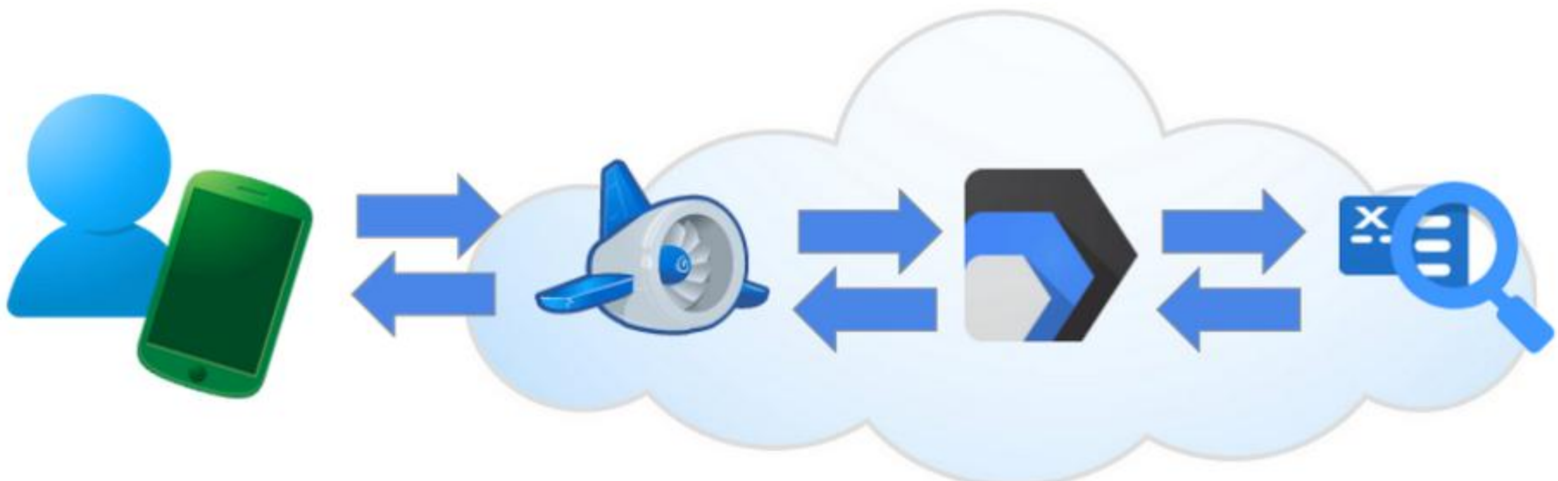
- Storage unstructured data in the cloud
- Individual objects can be huge - Terabyte+
- Stage raw CSV data before loading it into BigQuery

Workflow to help automate data transfer between these systems

App Engine Datastore	Web Scale collection of user data streams: a non-relational Datastore
Google Cloud Storage	Permanent archive of raw CSV data: cloud-based storage
Google BigQuery	Analysis of very large datasets

App Engine Pipeline API

- Provides a framework for automating workflows



Google Pipeline API

Define a Pipeline function

PYTHON

```python
class AddOne(pipeline.Pipeline):
  def run(self, number):
    return number + 1

add_pipeline = AddOne(1)
add_pipeline.start()                    #  Start the pipeline

pipeline_id = add_pipeline.pipeline_id    # Refer to the pipeline
```

PYTHON

```python
stage = AddOne.from_id(my_pipeline_id)
if stage.has_finalized:
  # do something with result
  print stage.outputs.default.value
```

Google Pipeline API

Connect two pipelines together

PYTHON

```
class AddTwo(pipeline.Pipeline):
  def run(self, number):
    result = yield AddOne(number)
    yield AddOne(result)
```

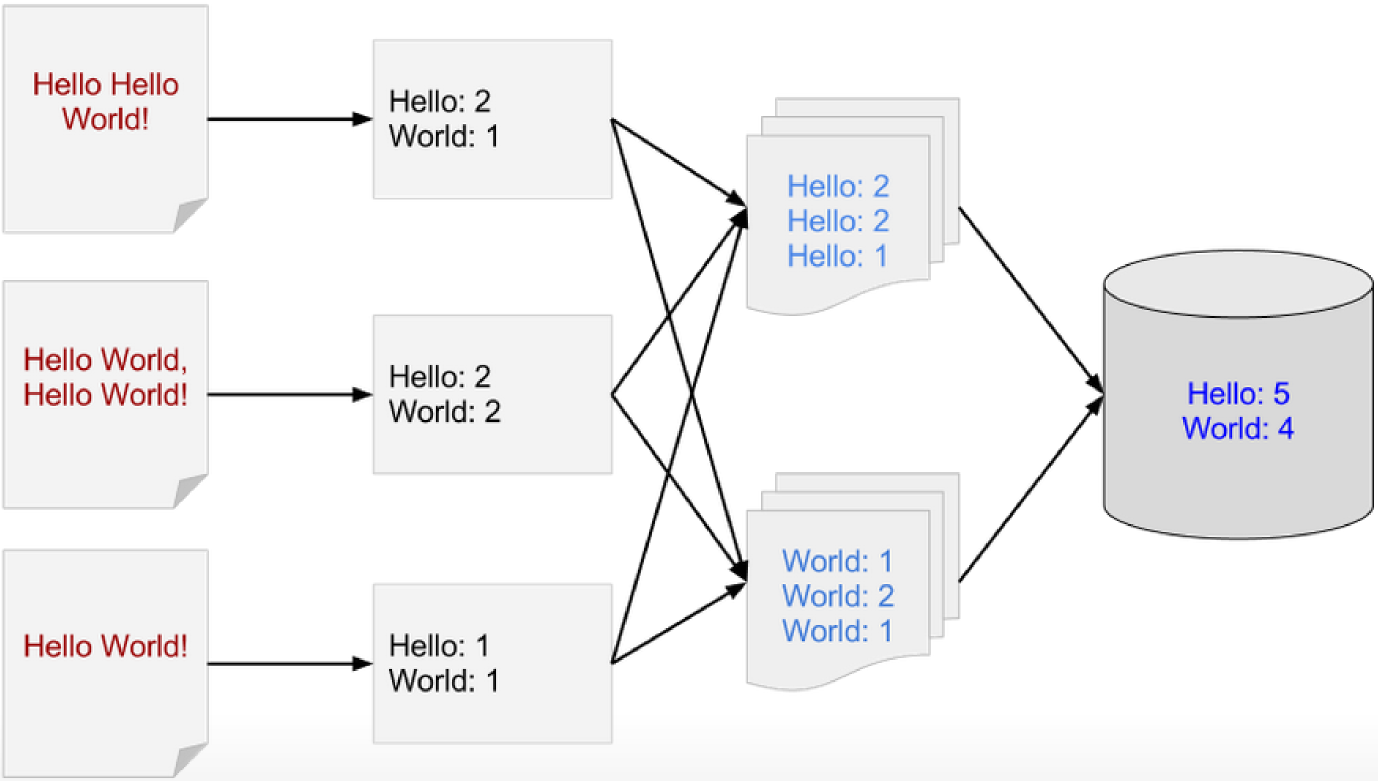
Pipeline Characteristics

- Pipeline patterns let you write procedural code
- Sources (inputs) and Sinks (outputs)
- Pipeline API: Provides a useful dashboard for viewing progress

MapReduce, of course!

- MapReduce is a powerful abstraction
- Allows for distributed processing over large datasets

Map Reduce Word Count: Reduce Phase



App Engine MapReduce

Focus on data transformation, not infrastructure details

- Open Source Library
- Built with App Engine Task Queues, BlobStore and the Pipeline API
- Provides input/output tools for many "sources and sinks"

App Engine MapReduce

PYTHON

```
class MyPipeline(base_handler.PipelineBase):
  def run(self, parameter):
    output = yield mapreduce_pipeline.MapreducePipeline(
        "name_of_pipeline_step",
        "main.map_function",                                  # A Mapper Function
        "main.reduce_function",                               # A Reduce Function
        "mapreduce.input_readers.DatastoreInputReader",       # Data Source
        "mapreduce.output_writers.FileOutputWriter",          # Data Sink
        mapper_params={},                                     # Custom Parameters for Mapper
        reducer_params={},                                    # Custom Parameters for Reducer
        shards=16)                                            # Workers per Job
    yield AnotherPipeline(output)
```

Why use Google App Engine?

- Scaling things yourself can be difficult/time consuming
- No need for new hardware infrastructure
- Avoid costs becoming unpredictable
- App Engine provides an excellent framework for app development

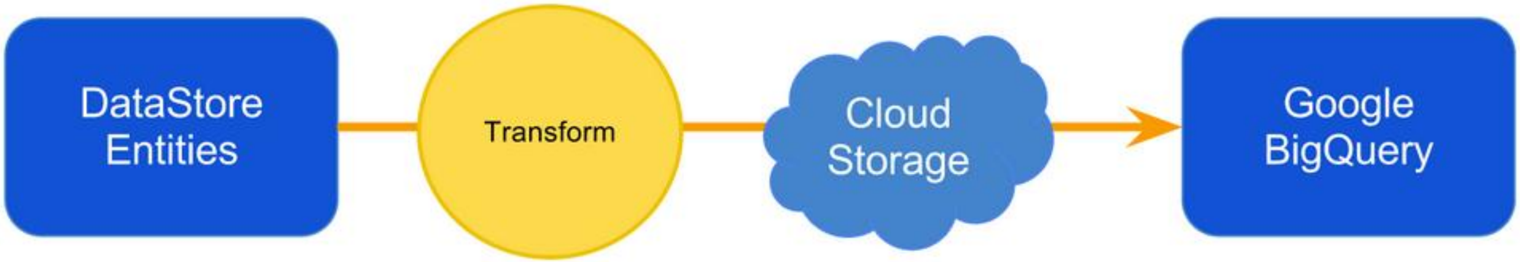
Data Pipelines in Practice

Examples and Code

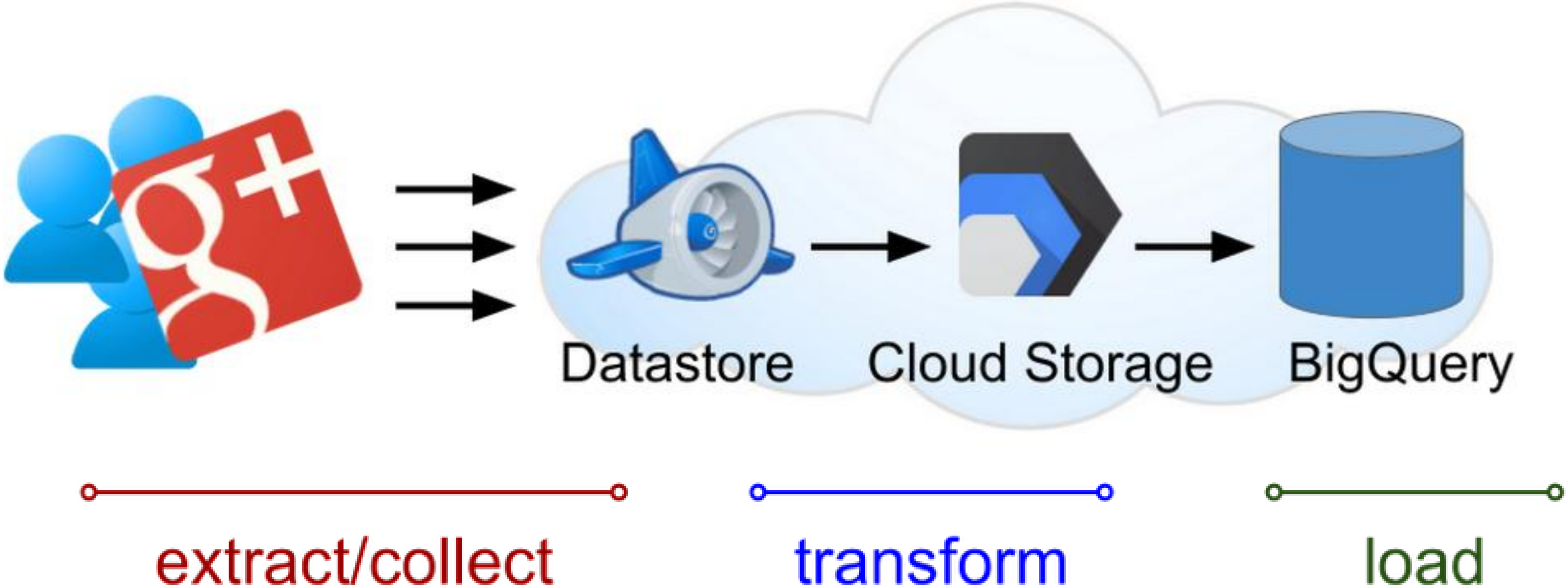
Example 1: Simple Datastore Mapper

Move Datastore Data into Google BigQuery

- Convert Datastore Data into CSV
- Simple Transformation (UTC Datestamp to Unix Epoch)



Example 1: Simple Datastore Mapper



Example 1: Simple Datastore Mapper

Iterate through Datastore, results into Cloud Storage

PYTHON

```python
class IteratorPipeline(base_handler.PipelineBase):
  def run(self, entity_type):
    output = yield mapreduce_pipeline.MapperPipeline(
      "Datastore_Iterator_" + entity_type,
      "main.datastore_map",
      "mapreduce.input_readers.DatastoreInputReader",
      output_writer_spec="mapreduce.output_writers.FileOutputWriter",
      params={
          "input_reader":{ "entity_kind": entity_type, },
          "output_writer":{
              "filesystem": "gs", "gs_bucket_name": GS_BUCKET, "output_sharding":"none",
              }
          },
          shards=SHARDS)
    yield CloudStorageToBigQuery(output)
```

Let's incorporate a simple transformation

Change Google+ timestamp to Unix Epoch Time

Let's incorporate a simple transformation

Change Google+ timestamp to Unix Epoch Time

BigQuery requires timestamps to be expressed in Unix epoch (integer)

PYTHON

```
import time, calendar

def convert_timestamp_to_epoch(timestamp):
  time_struct = time.strptime(timestamp, '%Y-%m-%dT%H:%M:%S.%fZ')
  return calendar.timegm(time_struct)
```

Example 1: Simple Datastore Mapper

Pipe resulting Cloud Storage Objects into BigQuery

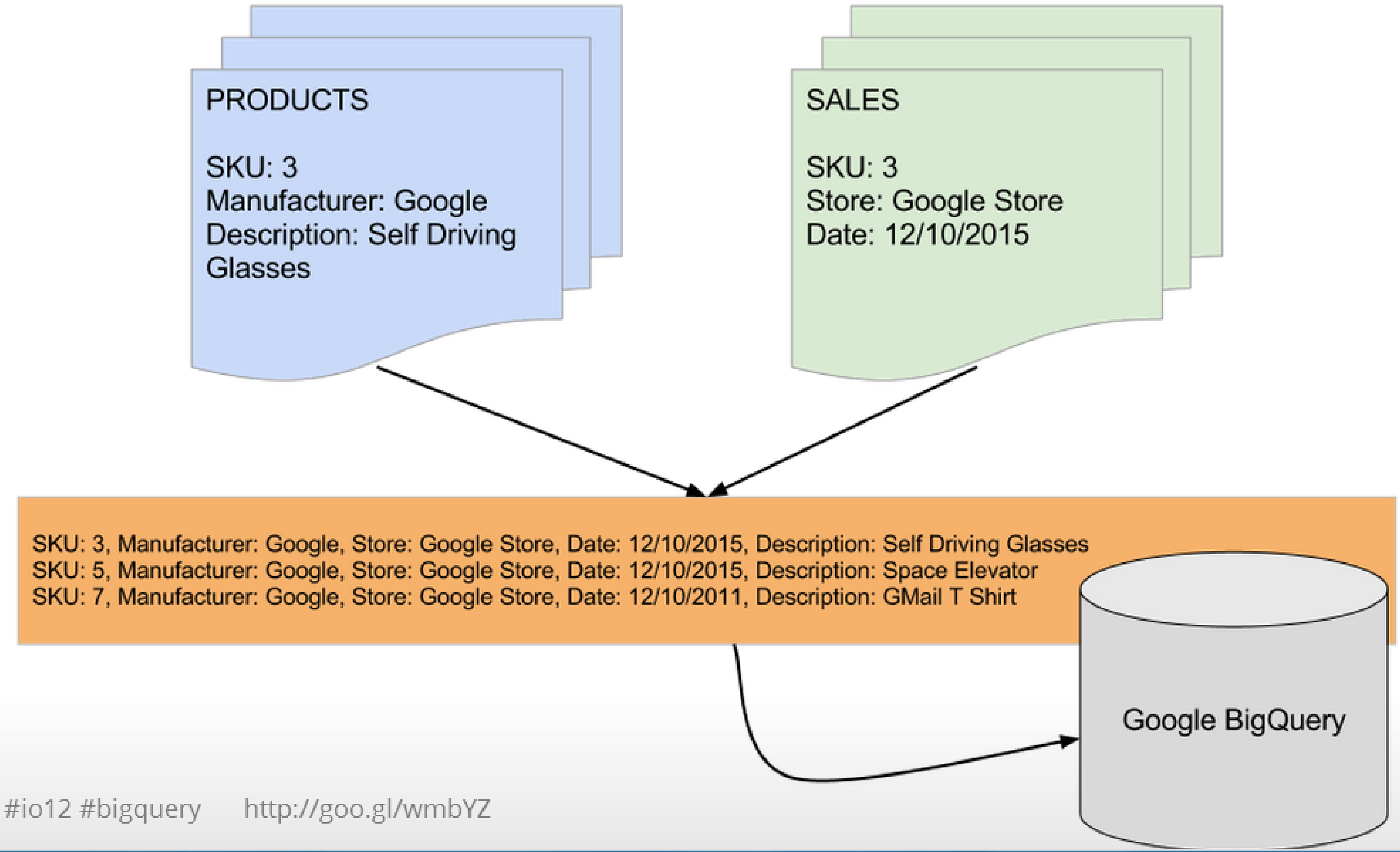
PYTHON

```python
class CloudStorageToBigQuery(base_handler.PipelineBase):
  def run(self, files):
    table_name = 'gplus_data_%s' % datetime.utcnow().strftime(
        '%m%d%Y_%H%M%S')
    jobs = bigquery_service.jobs()
    result = jobs.insert(projectId=PROJECT_ID,
                         body=build_job_data(table_name,files)).execute()
```

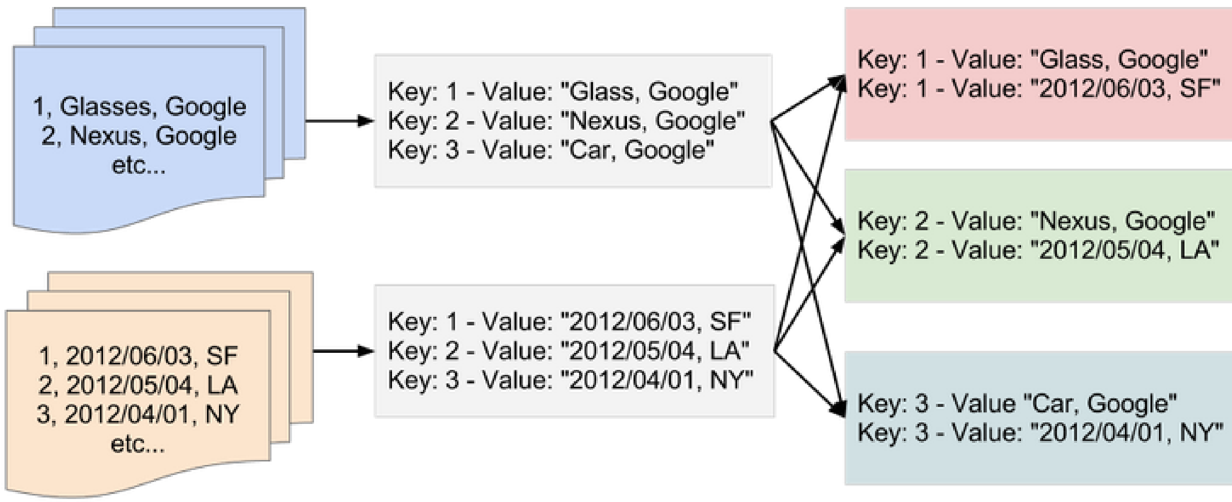
“Codelab: Querying App Engine logs with BigQuery ”

http://log2bq-codelab-io12.appspot.com/slides

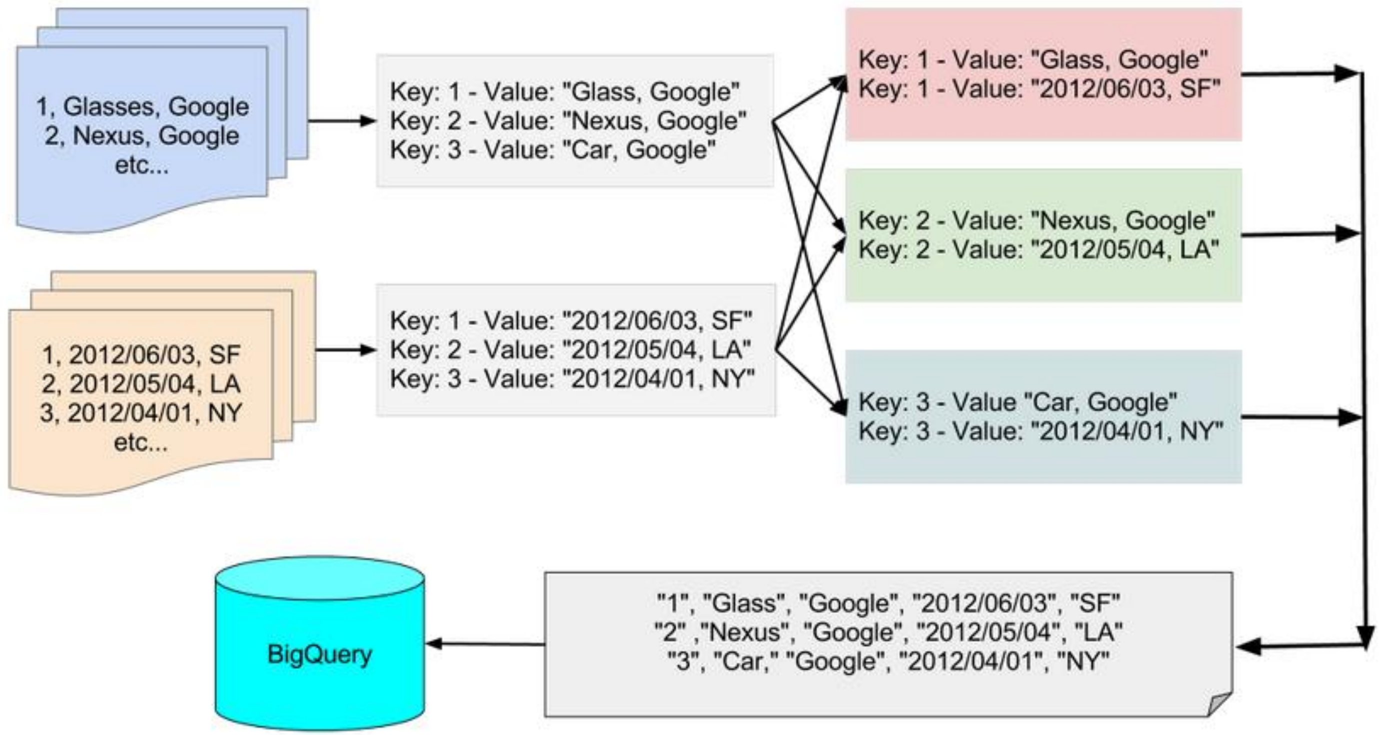
Example 2: JOIN Two Datastore Entities by Key



Step 2: Shuffle by Key



Step 3: Reduce - Join all Products/Sales per Key



Example 2: JOIN Two Datastore Entities by Key

PYTHON

```
class JoinOnSKU(base_handler.PipelineBase):
  def run(self):
    product_data = yield DatastoreMapperPipeline(
        mapper_spec = 'main.Datastore_map',
        entity_kind_spec = 'main.ProductDescription',
        shards=16)

    sales_data = yield DatastoreMapperPipeline(
        mapper_spec= 'main.Datastore_map',
        entity_kind_spec = 'main.ProductSales',
        shards=16)
```

Example 2: JOIN Two Datastore Entities by Key

PYTHON

```python
all_data = yield pipeline_common.Extend(product_data, sales_data)

shuffled_data = yield mapreduce_pipeline.ShufflePipeline(
    'Shuffle by Product ID',
    all_data,
    shards=16)
```

Example 2: JOIN Two Datastore Entities by Key

PYTHON

```
join_by_user_id = yield mapreduce_pipeline.ReducePipeline(
    'Join by SKU ID',
    'main.storage_reduce',
    output_writer_spec = 'mapreduce.output_writers.FileOutputWriter',
    params = {
        'output_writer':{
            'filesystem': 'gs',
                'gs_bucket_name': 'datastore_export',
                'output_sharding':'none'
              }
        },
    filenames = shuffled_data)
```

PYTHON

```
def storage_reduce(key, values):
  # Do something with the resulting values
  # A JOIN, a count, etc etc
  yield ('%s\n' % result)
```

Performance & Best Practices

Additional features you can adjust

- Number of Shards per Job
- App Engine Instance Size
- Task Queue Settings

BigShuffle: Experimental High Performance Shuffle

- Currently accepting a limited number of BigShuffle testers

Wrap Up:

- Build so that your computation is close to where your data lives
- Worry about your app, not your infrastructure
- Code: Keeps it simple, easy to maintain and test

<Thank You!>

Michael Manoochehri.

g+ gplus.to/manoochehri
twitter @nTangledMichael

github github.com/manoochehri