BIG DATA WITH PYTHON & GOOGLE BIGQUERY

github.com/jeffk/PyTexas-BigQuery

ME & THIS

- Jeff Kramer: github.com/jeffk & twitter.com/jeffk
- Work: Data Engineering at Vox Media
- Play: Late 80's to Early 00's Mac, SGI & NeXT Machines

- Presentation: GitHub.com/jeffk/PyTexas-BigQuery

WHAT IS BIG DATA?

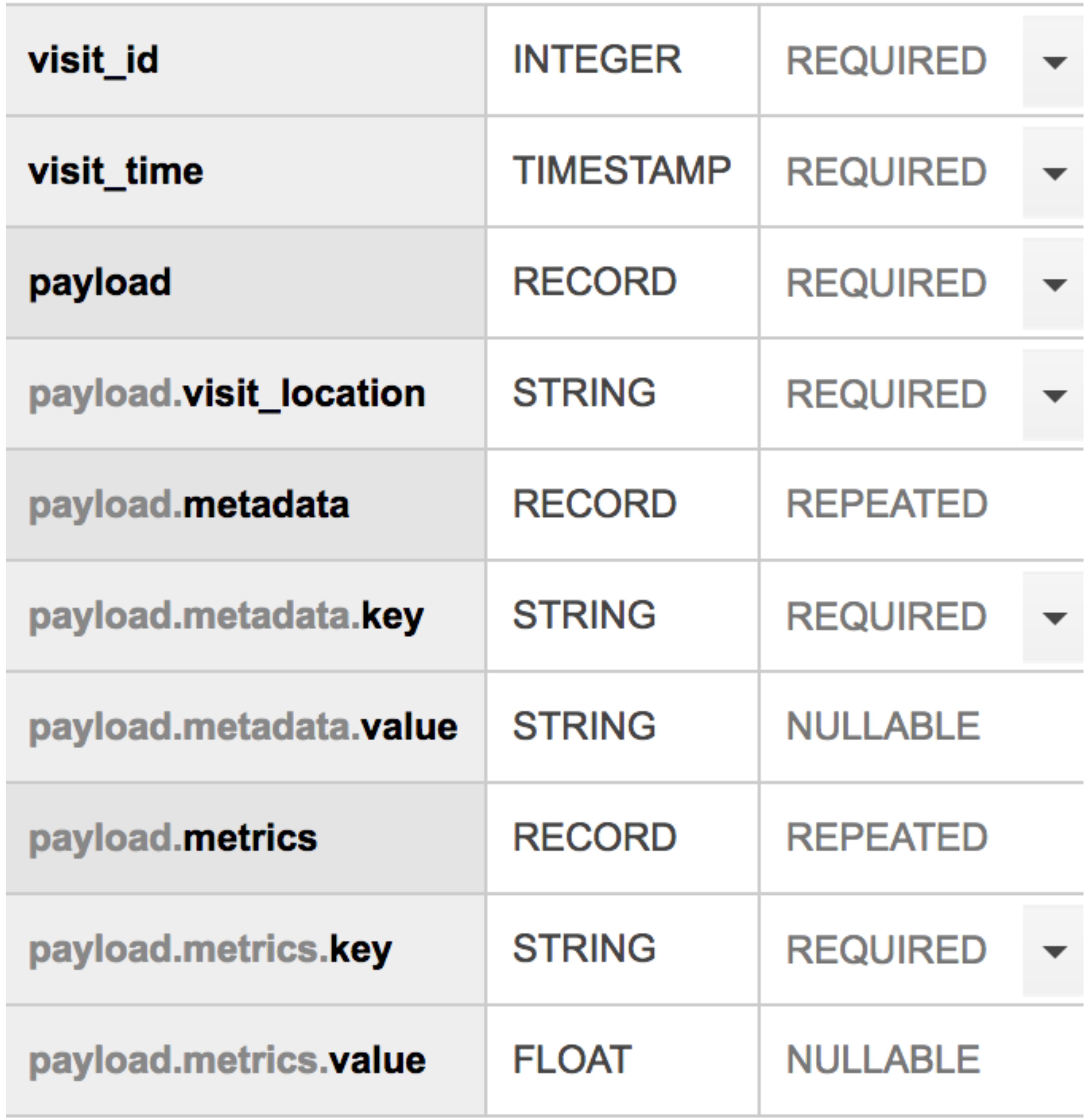
- It’s the new Data Warehouse
- Big Data is about Time, not about Space
- Problems bigger than a single core can handle must be distributed
- It’s really hard to do this well, and you have to make compromises
- BigQuery solves makes these compromises well for data archiving and analytics tasks

WHAT IS BIGQUERY?

- Google Cloud's SQL-Fluent Distributed OLAP Database
- Per-query pricing, cheap storage
- Converts your data into Google's custom Capacitor file format
- Fast at scale: SELECT … GROUP BY … across 100 Billion rows (3.6 TB) of Wikipedia logs in ~36 sec
- Loads/exports data through HTTP and bulk jobs

SUPPORTED DATA TYPES, QUERIES & ORGANIZATION

- SQL 2011: INT64, FLOAT64, BOOL, STRING, BYTES, DATE, DATETIME, TIME, TIMESTAMP, ARRAY, STRUCT
- STRUCTS == Nested || Repeated
- Standard SQL 2011 & Legacy BigQuery Extensions
- JavaScript UDFs
- Project -> Dataset -> Table



visit_id	INTEGER	REQUIRED
visit_time	TIMESTAMP	REQUIRED
payload	RECORD	REQUIRED
payload.visit_location	STRING	REQUIRED
payload.metadata	RECORD	REPEATED
payload.metadata.key	STRING	REQUIRED
payload.metadata.value	STRING	NULLABLE
payload.metrics	RECORD	REPEATED
payload.metrics.key	STRING	REQUIRED
payload.metrics.value	FLOAT	NULLABLE

CREATING DATASETS

```
client = bigquery.Client()
dataset_ref = client.dataset(name)
dataset = bigquery.Dataset(dataset_ref)
dataset.description = description
created_dataset = client.create_dataset(dataset)
```

CREATING SCHEMAS

```
SCHEMA = [
  SchemaField('visit_id', 'INT64',
    mode='required', description="Visit ID"),
  SchemaField('visit_time', 'TIMESTAMP',
    mode='required', description="Visit Time"),
  SchemaField('payload', 'STRUCT', mode='REQUIRED', fields = [
    SchemaField('visit_location', 'STRING',
      mode='required', description="Visit Location"),
    SchemaField('metadata', 'STRUCT', mode='REPEATED',fields = [
      SchemaField('key', 'STRING', mode='REQUIRED'),
      SchemaField('value', 'STRING')
    ]),
    SchemaField('metrics', 'STRUCT', mode='REPEATED', fields = [
      SchemaField('key', 'STRING', mode='REQUIRED'),
      SchemaField('value', 'FLOAT64')
    ])
  ])
]
```

CREATING TABLES

```
client = bigquery.Client()
table_ref = dataset.table(name)
table = bigquery.Table(table_ref, schema=SCHEMA)
table.description = description
created_table = client.create_table(table)
```

LOADING DATA VIA HTTP STREAMS

```
ROWS_TO_INSERT = [
    {'visit_id': 1,
    'visit_time': '2017-04-01T12:21:32',
    'payload':
        {'visit_location': 'NORTH',
        'metadata': [
            {‘key':'first_name', 'value':'Alice'},
            {‘key':'favorite_color', 'value':'red'},
            {‘key':'last_purchase_id', 'value':'1243'},
            {‘key':'last_purchase_total', 'value':'34.53'},
        ], 'metrics': [
            {‘key':'checkout_time', 'value': 82.4},
            {‘key':'net_promoter', 'value': 5},
            {‘key':'visit_count', 'value': 12},
        ]}
    }, …]
client = bigquery.Client()
errors = client.create_rows(table, ROWS_TO_INSERT)
```

QUERYING STRUCT DATA

```
client = bigquery.Client()
QUERY = """
SELECT visit_id, visit_time, payload.visit_location,
(SELECT value FROM UNNEST(payload.metadata) WHERE key = "first_name")
AS first_name,
(SELECT value FROM UNNEST(payload.metrics) WHERE key = "net_promoter")
AS net_promoter
FROM `%s.%s.%s` ORDER BY visit_id LIMIT 100
""" % (client.project, dataset_name, table_name)

rows = list(client.query_rows(QUERY, timeout=30))
for row in rows:
    print("%s\t%s\t%s\t%s\t%s" % (row[0], row[1], row[2], row[3], row[4]))
```

QUERYING DATA WITH A UDF

```
    QUERY = """
CREATE TEMPORARY FUNCTION rot13(x STRING)
RETURNS STRING
LANGUAGE js
AS \"\"\"
x = x.replace(/[a-zA-Z]/g,function(c){
   return String.fromCharCode((c<='Z'?90:122)\>=(c=c.charCodeAt(0)+13)?c:c-26);
});
return x;
\"\"\";
SELECT visit_id, visit_time, payload.visit_location,
  (SELECT rot13(value) FROM UNNEST(payload.metadata) WHERE key = "first_name")
    AS first_name,
  (SELECT value FROM UNNEST(payload.metrics) WHERE key = "net_promoter")
     AS net_promoter
FROM `%s.%s.%s` ORDER BY visit_id LIMIT 100
""" % (client.project, dataset_name, table_name)
```

RUN FANCY LOAD JOB

QUERYING INTO A TABLE

```
    client = bigquery.Client()
    QUERY = """
SELECT visit_id, visit_time, payload.visit_location,
  (SELECT value FROM UNNEST(payload.metadata) WHERE key = "first_name") AS first_name,
  (SELECT value FROM UNNEST(payload.metrics) WHERE key = "net_promoter")AS net_promoter
FROM `%s.%s.%s`
""" % (client.project, dataset_name, source_table)
    dataset = client.dataset(dataset_name)
    job_config = QueryJobConfig()
    job_config.destination = dataset.table(dest_table)
    job_config.write_disposition = 'WRITE_TRUNCATE'
    query_job = QueryJob(str(uuid.uuid4()), QUERY, client=client, job_config=job_config)
    query_job._begin()
    while not query_job.done():
        time.sleep(5)
    print("%s bytes processed." % query_job.total_bytes_billed)
```

EXPORTING A TABLE

```
client = bigquery.Client()
dataset = client.dataset(dataset_name)
table_ref = dataset.table(table)
job_config = ExtractJobConfig()
job_config.destination_format = 'AVRO'
dest = ['gs://%s/complex_query_output-*.avro' % bucket_name]
query_job = ExtractJob(str(uuid.uuid4()),
    table_ref, dest, client, job_config=job_config)
query_job._begin()
while not query_job.done():
    time.sleep(5)
if query_job.errors:
    print(query_job.errors)
print("%s file(s) created." %
  query_job._job_statistics().get('destinationUriFileCounts')[0])
```

LOADING DATA WITH A LOAD JOB: GOOGLE CLOUD STORAGE

```
client = bigquery.Client()
dataset_ref = client.dataset(dataset_name)
table_ref = dataset_ref.table(table+'$20171118')

job_config = bigquery.LoadJobConfig()
job_config.source_format = 'AVRO'
job_config._properties['timePartitioning'] = {'type': 'DAY'}
job_config.write_disposition = 'WRITE_TRUNCATE'

GS_URL = 'gs://{}/{}'.format(bucket_name, blob_name)
job = client.load_table_from_uri(
    GS_URL, table_ref, job_config=job_config)

while not job.done():
    time.sleep(5)
print('Loaded %s rows.' % job.output_rows)
```

LIMITING COST WITH PARTITIONED TABLES & PARTITION EXPIRATION

```
"""
SELECT visit_id, visit_time
FROM `%s.%s.%s`
WHERE _PARTITIONTIME BETWEEN
  TIMESTAMP('2017-11-01') AND TIMESTAMP('2017-11-15');
""" % (client.project, dataset_name, table_name)


table = bigquery.Table(table_ref, schema=SCHEMA)
table. partitioning_type = 'DAY'
created_table = client.create_table(table)
table.partition_expiration = 259200000 # 3 days
```

WHY BIGQUERY

- Super easy to run. It just works.
- Pay on demand: $5 per TB of data processed, $.05 per GB streamed in
- Storage costs are low: $.02 per gig per month, $.01 if not edited in last 90 days
- Easy connections to other Google products (Google Analytics Data, YouTube data, Data Studio, Cloud Dataflow/Apache Beam, Google Sheets, etc)

WHY NOT BIGQUERY

- Application-style workloads (web site, OLTP): Just use PostgreSQL
- Vendor lock-in: You may be in AWS already, you may be on-premises and need something like Apache Impala
- Limits on table UPDATE/DELETEs: 100 per table per day