This presentation is out of date. See the following link for more up to date information on using Apache Cassandra from Python.

https://speakerdeck.com/tylerhobbs/intro-to-cassandra-and-the-python-driver

Using Apache Cassandra from Python

Jeremiah Jordan

Morningstar, Inc.

MORNINGSTAR®

Who am I?

Software Engineer @ Morningstar, Inc. for 1.5 Years

Using Python 2.6/2.7 for 1.5 Years

Using Cassandra for 1.5 Years

Why are you here?

You were too lazy to get out of your seat.

Someone said “NoSQL”.

You want to learn about using Cassandra from Python.

What am I going to talk about?

What is Cassandra

Starting up a local dev/unit test instance

Using Cassandra from Python

Indexing / Schema Design

What am I not going to talk about?

Setting up and maintaining a production cluster

Where can I get the slides?

http://goo.gl/8Byd8

points to

http://www.slideshare.net/jeremiahdjordan/pycon-2012-apache-cassandra

What is Apache Cassandra? (Buzz Word Description)

"Cassandra is a highly scalable, eventually consistent, distributed, structured key-value store. Cassandra brings together the distributed systems technologies from Dynamo and the data model from Google's BigTable. Like Dynamo, Cassandra is eventually consistent. Like BigTable, Cassandra provides a ColumnFamily-based data model richer than typical key/value systems."

From the Cassandra Wiki: http://wiki.apache.org/cassandra/

What is Apache Cassandra?

Column based key-value store (multi-level dictionary)

Combination of Dynamo (Amazon)and BigTable (Google)

Schema-optional

Keyspace 1

Column Family 1

Key1	ColumnName1	ColumnName2	ColumnName3	ColumnName4
	Value1	Value2	Value3	Value4
Key2	ColumnName1	ColumnName2	ColumnName3	ColumnName4
	Value1	Value2	Value3	Value4

Column Family 2

Key1	ColumnName1	ColumnName2	ColumnName3	ColumnName4
	Value1	Value2	Value3	Value4

Keyspace 2

Keyspace: "ApplicationData"

Column Family: "UserInfo"

John	age	email	gender	state
	32	john@gmail.com	M	IL
Bill	email	gender	state	
	bill@apache.org	M	CA	
Jane	age	email	state	
	25	jane@python.org	VA	

Column Family: "ChangesOverTime"

2012-03-01	time-uuid-1	time-uuid-2	time-uuid-3	time-uuid-4
	John:state::CA	Bill:gender:F:M	John:state:CA:IL	Jane:age::25

Keyspace: "OtherData"

Keyspace: "ApplicationData"

Column Family: "UserInfo"

Keys

John	age	email	gender	state
	32	john@gmail.com	M	IL
Bill	email	gender	state	
	bill@apache.org	M	CA	
Jane	age	email	state	
	25	jane@python.org	VA	

Column Family: "ChangesOverTime"

2012-03-01	time-uuid-1	time-uuid-2	time-uuid-3	time-uuid-4
	John:state::CA	Bill:gender:F:M	John:state:CA:IL	Jane:age::25

Keyspace: "OtherData"

Keyspace: "ApplicationData"

Column Family: "UserInfo"

Column Names →

John	age	email	gender	state
	32	john@gmail.com	M	IL
Bill	email	gender	state	
	bill@apache.org	M	CA	
Jane	age	email	state	
	25	jane@python.org	VA	

Column Family: "ChangesOverTime"

2012-03-01	time-uuid-1	time-uuid-2	time-uuid-3	time-uuid-4
	John:state::CA	Bill:gender:F:M	John:state:CA:IL	Jane:age::25

Keyspace: "OtherData"

Keyspace: "ApplicationData"

Column Family: "UserInfo"

Column Values

John	age	email	gender	state
	32	john@gmail.com	M	IL
Bill	email	gender	state	
	bill@apache.org	M	CA	
Jane	age	email	state	
	25	jane@python.org	VA	

Column Family: "ChangesOverTime"

2012-03-01	time-uuid-1	time-uuid-2	time-uuid-3	time-uuid-4
	John:state::CA	Bill:gender:F:M	John:state:CA:IL	Jane:age::25

Keyspace: "OtherData"

Keyspace: "ApplicationData"

Column Family: "UserInfo"

Timestamps

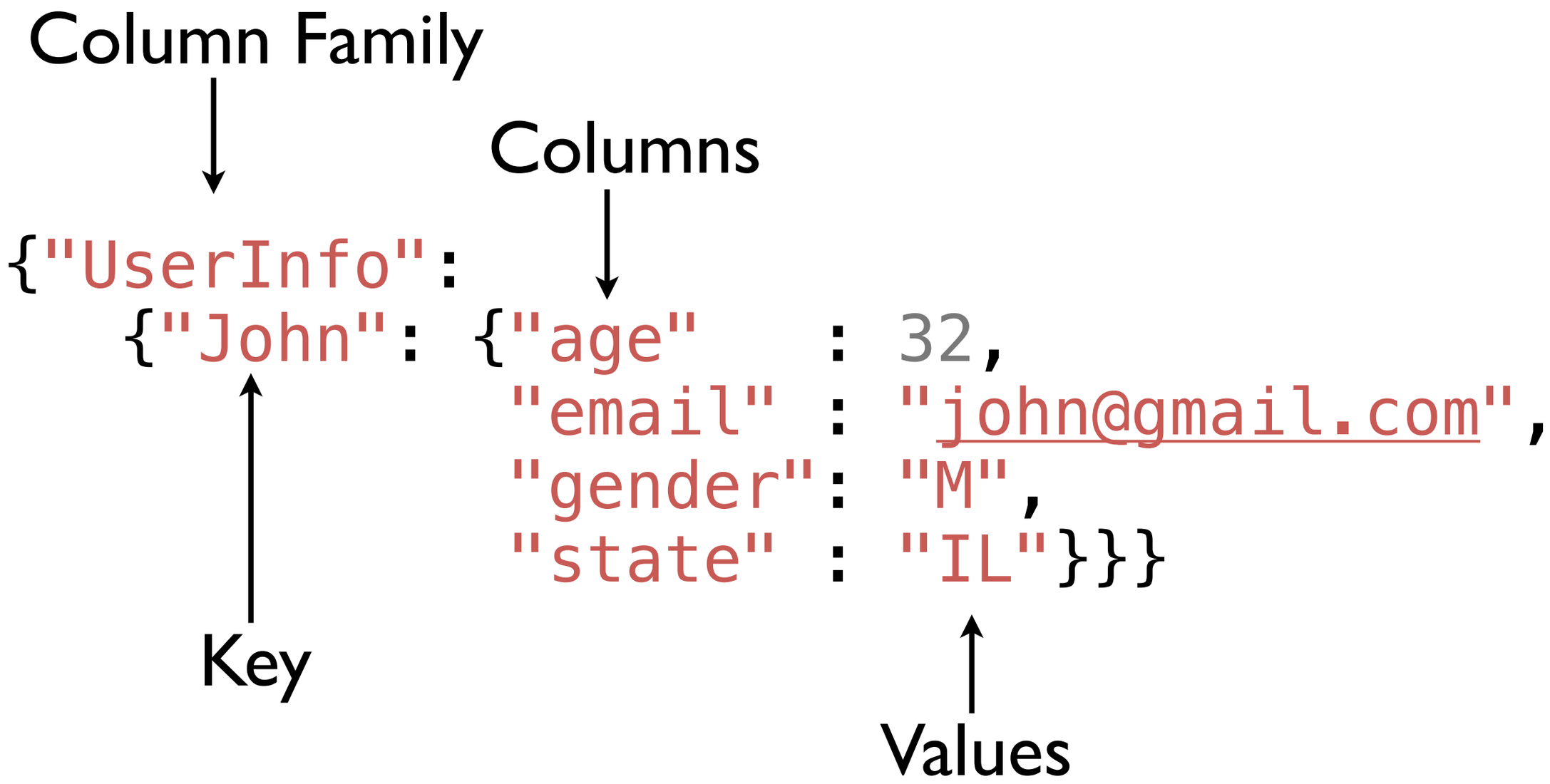
John	age	email	gender	state
	32 2012-03-08T12:00:01.56	john@gmail.com 2012-03-08T12:00:01.56	M 2012-03-08T12:00:01.56	IL 2012-03-01T13:50:01.56
Bill	email	gender	state	
	bill@apache.org 2012-03-07T12:00:01.56	M 2012-03-01T13:00:01.56	CA 2012-03-08T12:00:01.42	
Jane	age	email	state	
	25 2012-03-01T15:00:01.56	jane@python.org 2012-03-08T12:00:01.56	VA 2012-03-08T12:00:01.56	

Column Family: "ChangesOverTime"

2012-03-01	time-uuid-1	time-uuid-2	time-uuid-3	time-uuid-4
	John:state::CA 2012-03-01T12:00:01.56	Bill:gender:F:M 2012-03-01T13:00:01.56	John:state:CA:IL 2012-03-01T13:50:01.56	Jane:age::25 2012-03-01T15:00:01.56

Keyspace: "OtherData"

Multi-level Dictionary



Well really this

```
{"UserInfo":
    {"John":
        OrderedDict(
            [("age", 32),
             ("email", "john@gmail.com"),
             ("gender", "M"),
             ("state", "IL")])}}
```

Where do I get it?

From the Apache Cassandra project:

http://cassandra.apache.org/

Or DataStax hosts some Debian and RedHat packages:

http://www.datastax.com/docs/1.0/install/

How do I run it?

Edit conf/cassandra.yaml

Change data/commit log locations

defaults: /var/cassandra/data and /var/cassandra/commitlog

Edit conf/log4j-server.properties

Change the log location/levels

default: /var/log/cassandra/system.log

How do I run it?

Edit conf/cassandra-env.sh
(bin/cassandra.bat on windows)

Update JVM Memory usage

default: 1/2 your ram

How do I run it?

Foreground

```
$ ./cassandra -f
```

Setup tips for local instances

Make templates out of cassandra.yaml and log4j-server.properties

Update “cassandra” script to generate the actual files (run them through “sed” or something)

Server is running, what now?

```
$ ./cassandra-cli
connect localhost/9160;

create keyspace ApplicationData
with placement_strategy =
 'org.apache.cassandra.
            locator.SimpleStrategy'
and strategy_options =
          [{replication_factor:1}];
```

Server is running, what now?

```
use ApplicationData;

create column family UserInfo
    and comparator = 'AsciiType';

create column family ChangesOverTime
    and comparator = 'TimeUUIDType';
```

Connect from Python

http://wiki.apache.org/cassandra/ClientOptions

Thrift - See the “interface” directory (Do not use!!!)

Pycassa - pip install pycassa

Telephus (twisted) - pip telephus

DB-API 2.0 (CQL) - pip cassandra-dbapi2

Thrift (don't use it)

```
from thrift.transport import TSocket, TTransport
from thrift.protocol import TBinaryProtocol
from pycassa.cassandra.c10 import Cassandra, ttypes

socket = TSocket.TSocket('localhost', 9160)
transport = TTransport.TFramedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocolAccelerated(transport)
client = Cassandra.Client(protocol)
transport.open()
client.set_keyspace('ApplicationData')
import time
client.batch_mutate(
  mutation_map=
      {'John': {'UserInfo':
               [ttypes.Mutation(
                  ttypes.ColumnOrSuperColumn(
                    ttypes.Column(name='email',
                                  value='john@gmail.com',
                                  timestamp= long(time.time()*1e6),
                                  ttl=None)))]}},
  consistency_level= ttypes.ConsistencyLevel.QUORUM)
```

Pycassa

```
import pycassa
from pycassa.pool import ConnectionPool
from pycassa.columnfamily import ColumnFamily

pool = ConnectionPool('ApplicationData',
                      ['localhost:9160'])
col_fam = ColumnFamily(pool, 'UserInfo')
col_fam.insert('John', {'email': 'john@gmail.com'})
```

Pycassa

http://pycassa.github.com/pycassa/

https://github.com/twissandra/twissandra

Connect

Keyspace

```
pool = ConnectionPool('ApplicationData',
                      ['localhost:9160'])
```

Server List

Open Column Family

Connection Pool

```
col_fam = ColumnFamily(pool,
                       'UserInfo')
```

Column Family

Write

Key

Column Value

```
col_fam.insert('John',
               {'email':
                    'john@gmail.com'})
```

Column Name

Read

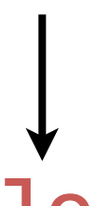
```
readData = col_fam.get('John',
                       columns=['email'])
```

Key

Column Names

Delete

Key



```
col_fam.remove('John',
               columns=['email'])
```

Column Names

Batch

Column Names

```
col_fam.batch_insert(
    {'John': {'email': 'john@gmail.com',
              'state': 'IL',
              'gender': 'M'},
'Jane': {'email': 'jane@python.org',
              'state': 'CA'}})
```

Keys

Column Values

Batch (streaming)

```
b = col_fam.batch(queue_size=10)
b.insert('John',
         {'email': 'john@gmail.com',
          'state': 'IL',
          'gender': 'F'})

b.insert('Jane',
         {'email': 'jane@python.org',
          'state': 'CA'})
```

Batch (streaming)

```
b.remove('John', ['gender'])
b.remove('Jane')
b.send()
```

Batch (Multi-CF)

```
from pycassa.batch import Mutator
import uuid
b = Mutator(pool)
b.insert(col_fam,
         'John', {'gender':'M'})
b.insert(index_fam,
         '2012-03-09',
         {uuid.uuid1().bytes:
                  'John:gender:F:M'})
```

Batch Read

```
readData = col_fam.multiget(['John',
                             'Jane',
                             'Bill'])
```

Column Slice

```
d = col_fam.get('Jane',
                column_start='email',
                column_finish='state')

d = col_fam.get('Bill',
                column_reversed=True,
                column_count=2)
```

Column Slice

```
startTime = pycassa.util.
    convert_time_to_uuid(time.time()-600)

d = index_fam.get('2012-03-31',
                  column_start=startTime,
                  column_count=30)
```

Types

```
from pycassa.types import *
col_fam.column_validators['age'] =
    IntegerType()
col_fam.column_validators['height'] =
    FloatType()

col_fam.insert('John', {'age':32,
                        'height':6.1})
```

Column Family Map

```
from pycassa.types import *
class User(object):
    key = Utf8Type()
    email = AsciiType()
    age = IntegerType()
    height = FloatType()
    joined = DateType()
```

Column Family Map

```
from pycassa.columnfamilymap import
    ColumnFamilyMap
cfmap = ColumnFamilyMap(User, pool,
                        'UserInfo')
```

Write

```
from datetime import datetime
user = User()
user.key = 'John'
user.email = 'john@gmail.com'
user.age = 32
user.height = 6.1
user.joined = datetime.now()
cfmap.insert(user)
```

Read/Delete

```
user = cfmap.get('John')
```

```
users = cfmap.multiget(['John', 'Jane'])
```

```
cfmap.remove(user)
```

Timestamps/Consistency

```
col_fam.read_consistency_level =
    ConsistencyLevel.QUORUM
col_fam.write_consistency_level =
    ConsistencyLevel.ONE
```

```
col_fam.get('John',
            read_consistency_level=
                ConsistencyLevel.ONE)

col_fam.get('John',
            include_timestamp=True)
```

Indexing

Native secondary indexes

Roll your own with wide rows

Indexing Links

Intro to indexing

http://www.datastax.com/dev/blog/whats-new-cassandra-07-secondary-indexes

Blog post and presentation going through some options

http://www.anuff.com/2011/02/indexing-in-cassandra.html

http://www.slideshare.net/edanuff/indexing-in-cassandra

Another blog post describing different patterns for indexing

http://pkghosh.wordpress.com/2011/03/02/cassandra-secondary-index-patterns/

Native Indexes

Easy to add, just update the schema

Can use filtering queries

Not recommended for high cardinality values (i.e. timestamps, birth dates, keywords, etc.)

Makes writes slower to indexed columns (read before

Add Index

```
update column family UserInfo
  with column_metadata=[
        {column_name: state,
         validation_class: UTF8Type,
         index_type: KEYS};
```

Native Indexes

```
from pycassa.index import *
state_expr = create_index_expression('state',
                                     'IL')
age_expr = create_index_expression('age',
                                   20,
                                   GT)
clause = create_index_clause([state_expr,
                              age_expr],
                             count=20)
for key, userInfo in \
        col_fam.get_indexed_slices(clause):
    # Do Stuff
```

Rolling Your Own

Removing changed values yourself

Know the new value doesn’t exists, no read before write

Index can be denormalized query, not just an index.

Can use things like composite columns, and other tricks to

Lessons Learned

Use indexes. Don’t iterate over keys.

New Query == New Column Family

Don’t be afraid to write your data to multiple places (Batch)

Questions?