

# Using Apache Cassandra from Python

Jeremiah Jordan

Morningstar, Inc.



# 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**

**Using Cassandra from Python**

**Indexing**

**Other stuff to consider**

# **What am I not going to talk about?**

**Setting up and maintaining a production Cassandra cluster.**

# **What is Apache Cassandra?**

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

**Combination of Dynamo (Amazon)and BigTable (Google)**

**Schema-optimal**

# 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/>

# Keyspace 1

## Column Family 1

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

## Column Family 2

	ColumnName1	ColumnName2	ColumnName3	ColumnName4
Key1	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	
	20		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	
		20	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
		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	
	20	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

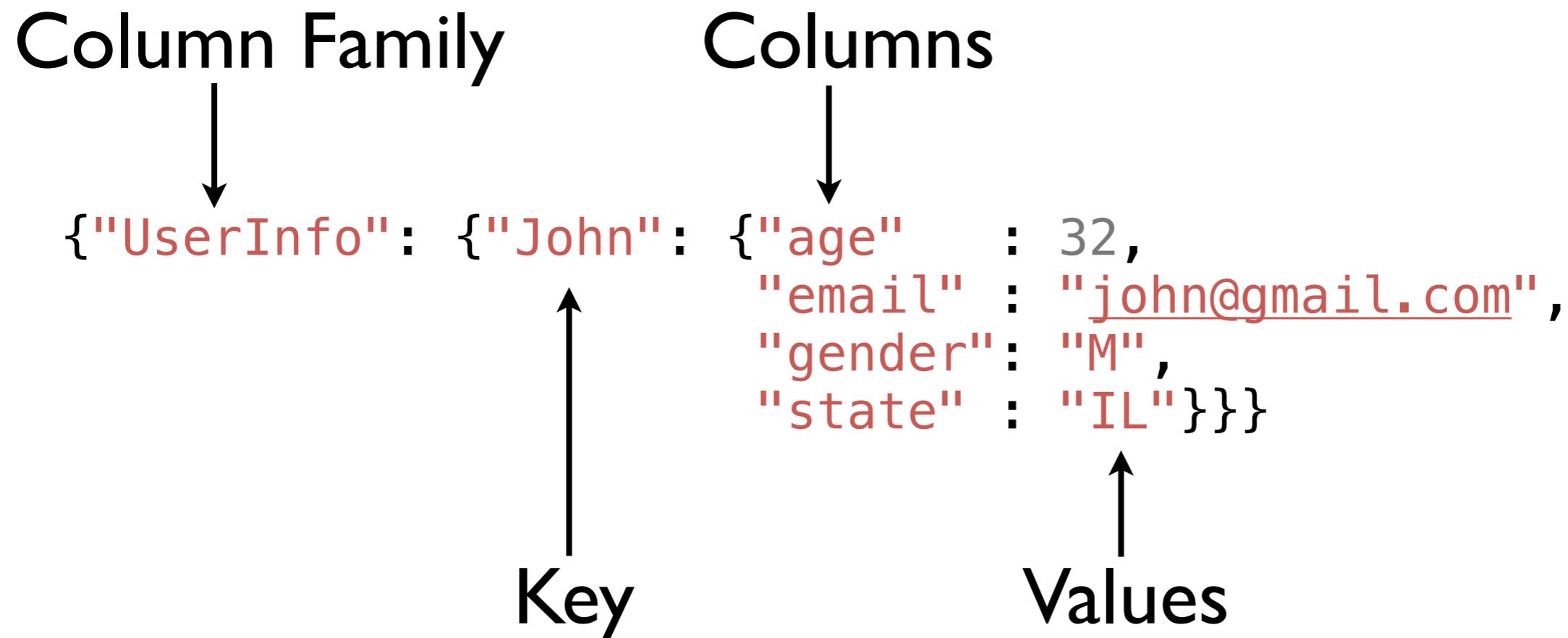
John	age	email	gender	state
	32	john@gmail.com	M	IL
Bill	email	gender	state	
	20	M	CA	
Jane	age	email	state	
	25	jane@python.org	VA	

## 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”

# 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/install\\_package](http://www.datastax.com/docs/1.0/install/install_package)

# How do I run it?

**Edit cassandra.yaml**

**Change data/commit log locations**

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

**Edit log4j-server.properties**

**Change the log location/levels**

**default /var/log/cassandra/system.log**

# How do I run it?

Foreground



\$ ./cassandra -f

# **Setup tip 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?

```
connect localhost/9160;

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

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

Pycassa - pip/easy\_install pycassa

Telephus (twisted) - pip/easy\_install telephus

DB-API 2.0 (CQL) - pip/easy\_install 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'})
```

# Connect

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

```
graph TD; Keyspace[Keyspace] --> Pool(pool = ConnectionPool('ApplicationData', ['localhost:9160'])); ServerList[Server List] --> Servers(['localhost:9160']);
```

# Open Column Family

Connection Pool



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

The code snippet shows the creation of a ColumnFamily object named 'col\_fam' using a connection pool and specifying the column family name as 'UserInfo'. The word 'UserInfo' is highlighted in red.

Column Family

# Write

Key                          Column Value

↓                            ↓

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

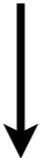
↑                            Column Name

# Read

```
Key  
↓  
readData = col_fam.get('John',  
                      columns=['email'])  
↑  
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'})

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'})

b.send()
```

# Batch Read

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

# Column Slice

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

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

# 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 = Utf8Type()
    age = IntegerType()
    height = FloatType()
    joined = DateType()

from pycassa.columnfamiliemap 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)
```

# **Indexing**

**Native secondary indexes**

**Roll your own with wide rows**

**Composite Columns**

# 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 use

Let you use filtering queries

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

Make writes slower to indexed columns (read before write)

# 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,
                               bday_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 allow range like searches.

# **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)**