Getting Started with Riak TS

Overview

Welcome to the Getting Started Guide for Riak TS. This guide provides instructions to walk you through your first project with Riak TS. Basho created a Sandbox environment to help you quickly get started learning about Riak TS without having to worry about installing and configuring a Riak TS cluster. For those who want to know how Sandbox was implemented, links are provided in the **Additional Information** section at the end of this document.

This sandbox, as the name implies, is meant to be an environment where you can quickly learn to use Riak TS. It is NOT meant to be a production environment or represent what a production environment should look like.

The Sandbox is configured to run 3 instances of Riak TS in a cluster. Each node will run in Virtual Box with Ubuntu 14.04, 2 GB RAM, 2 CPUs. Vagrant box is the container for this cluster environment.

By default, this demo will:

- Provision 3 VMs running Ubuntu 14.04, each with 2GB RAM and 2 CPU
- Install Riak TS via the ansible-riak ansible role
- Cluster Riak TS on all VMs
- Configure Riak Shell on all nodes for use with the cluster
- Create the Python virtualenv in the demo directory, `/home/vagrant/ts-demo` where Riak Python client, pandas and matplotlib are installed.
- Set up a demo in `/home/vagrant/ts-demo` that includes:
    - An IPython notebook for creating Time Series tables
    - About 1 million rows of real world Time Series data and a loader script
    - An IPython notebook to query and manipulate data within Riak TS
- Allows you to launch the Jupyter webserver from your host environment to run the IPython notebooks
- The servers are `riak-ts1`, `riak-ts2`, and `riak-ts3`.

To install and build the Sandbox environment, the following software must be installed on your machine.

- git (Tested with 1.9.1)
- ansible (Tested with 2.0.1.0)
- vagrant (Tested with 1.8.1)
- virtualBox (Tested with 5.0.16)

You must also have an HTML5 supported browser for Jupyter (IPython version 3) notebooks.

Introduction to Riak TS

Riak TS was purposely built to handle Time Series and Internet of Things (IoT) use cases that are wide ranging and growing quickly as new technologies and device proliferation are producing more data. Some ways Time Series is being used can be found here.

An enterprise time series database is purposely built to collect, store, manage and analyze data at scale, allowing you to focus on getting the most value from your data to improve the way your organization does business. It allows you to gather large amounts of time based information and then query the data to answer questions such as during what time of day do most people buy groceries, or spanning a larger timeframe by asking during what times of the week do people buy groceries? What times do the fewest people buy groceries, which might change the hours a store stays open, allowing you to save time and money.

In order to provide faster and easier queries, Riak TS stores your data in blocks called a **quantum** (quanta is the plural of quantum), which segments your data in a manner that makes sense for your data. You might choose to store data based on the **city** the store is located in, and then **store number**, for each store within a given city. Or it might make sense configure for regions and cities.

By storing data in quanta, when you query to answer business questions, getting your data from fewer servers in larger chunks will make your queries faster. Using familiar SQL commands to answer your questions allows business analysts to formulate the questions and find the answers.

Riak TS is built on the same foundation as Riak KV, known for its high availability, resilience, fault tolerance, horizontal scalability using commodity hardware, and simplicity of operational management. For more details on how Riak TS works, read the blog by one of our Solution Architects here.

Aarhus Demo

The Aarhus demo uses a collection of sensor readings from Aarhus, the second largest city in Denmark over a period of 5 months, collected from February to June 2014. The data was gathered as planning for a Smart City Framework where services to improve everyday life are offered to citizens. The complete dataset includes information on pollution, weather, cultural events, library events parking and traffic. You know that collecting and understanding environmental data would provide valuable insights for city planners. Today, you will take a closer look at the traffic information gathered for Aarhus.

The traffic data was gathered from sensors along the road, recording the time it takes traffic to pass between two sensors. The data stored is:

status of the sensor

Average Measured Time to pass between the sensors.

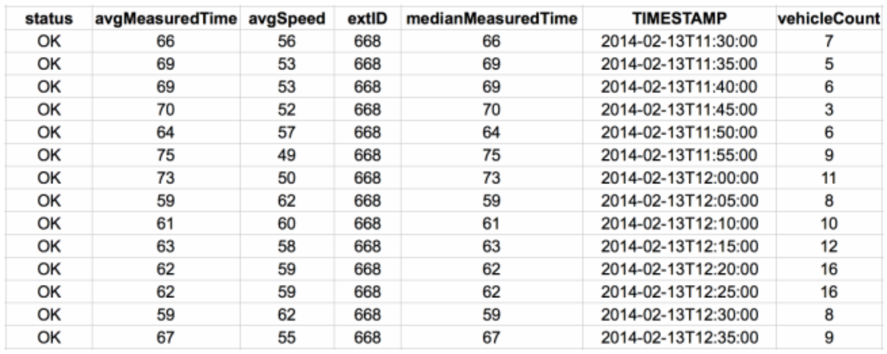
Average Speed in km/h

extID, External ID of the sensor pair.

Median Measured Time to pass between the sensors

Timestamp captured in 5 minute intervals,

Vehicle count, the number of vehicles passing between the 2 sensors during the 5 minute interval



status	avgMeasuredTime	avgSpeed	extID	medianMeasuredTime	TIMESTAMP	vehicleCount
OK	66	56	668	66	2014-02-13T11:30:00	7
OK	69	53	668	69	2014-02-13T11:35:00	5
OK	69	53	668	69	2014-02-13T11:40:00	6
OK	70	52	668	70	2014-02-13T11:45:00	3
OK	64	57	668	64	2014-02-13T11:50:00	6
OK	75	49	668	75	2014-02-13T11:55:00	9
OK	73	50	668	73	2014-02-13T12:00:00	11
OK	59	62	668	59	2014-02-13T12:05:00	8
OK	61	60	668	61	2014-02-13T12:10:00	10
OK	63	58	668	63	2014-02-13T12:15:00	12
OK	62	59	668	62	2014-02-13T12:20:00	16
OK	62	59	668	62	2014-02-13T12:25:00	16
OK	59	62	668	59	2014-02-13T12:30:00	8
OK	67	55	668	67	2014-02-13T12:35:00	9

In this Getting Started, you will:

- Use an Jupyter (IPython v3) Notebooks to create the aarhus2 table in Riak TS
- Load the sensor data using a command line Python script
- Query the database to answer questions.
- Use riak-shell, the command line interface to execute SQL commands
- Use pandas to work with data frames.
- Use matplotlib to display a graph of the data

Let's Get Started

It is assumed your environment is configured so you can clone a GitHub repository, and you have installed git, ansible, Virtual Box and Vagrant as listed above.

Clone the GitHub Repository

1. From the command line window, clone the GitHub repository.

```
git clone https://github.com/basho-labs/riak-demos.git
```

2. Change to the riak-demos directory

```
cd riak-demos
```

Initialize And Pull Down The Ansible-Riak Role

3. Initialize the submodule

```
git submodule init
```

4. Update the module

```
git submodule update
```

5. Spin up the demo cluster

   ```
   vagrant up
   ```

   Note: This will take a while to download Ubuntu and build your environment. You will know the process has completed when you see the command prompt return and no error messages are listed above.

Launch the Jupyter Notebook

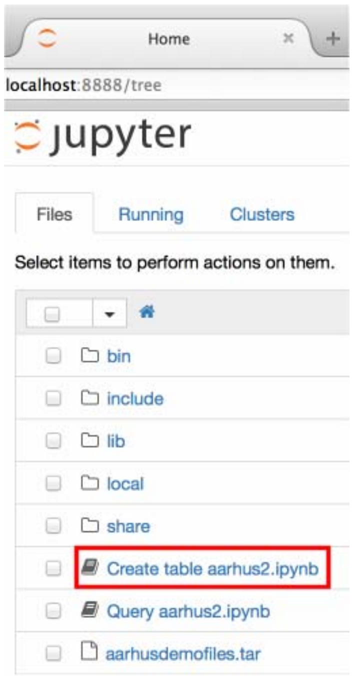
6. Launch the Jupyter Notebook from the command line of your host environment.

   ```
   vagrant ssh riak-ts1 -- -NfgL8888:localhost:8888
   ```

7. Bring up Jupyter environment in your (HTML5 compatible) Browser on your host computer.

   ```
   http://localhost:8888
   ```

8. Open the Create table aarhus2 Notebook by double clicking on the link.

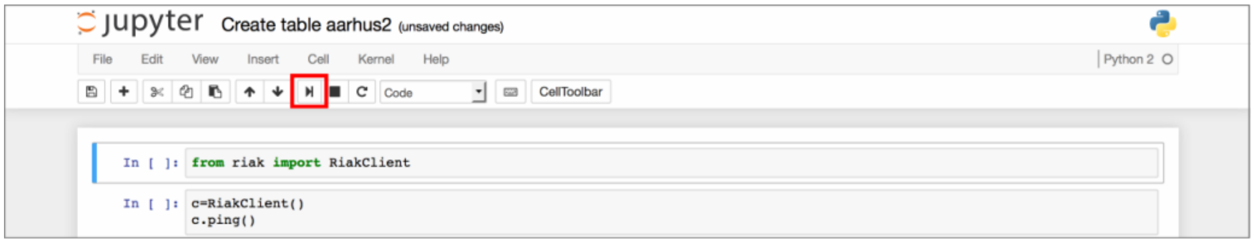


Select items to perform actions on them.

bin
include
lib
local
share
Create table aarhus2.ipynb
Query aarhus2.ipynb
aarhusdemofiles.tar

Each cell contains Python code that will run in real time when you click the run button. If you took the Python code in each of the cells and put it in a text file, you could run this as a Python file from the command line. Notebooks are a great way to execute code in small increments and observe the outcome while you do development. It also allows you to easily share your code with the rest of your team so they are able execute your code in the way you intended.

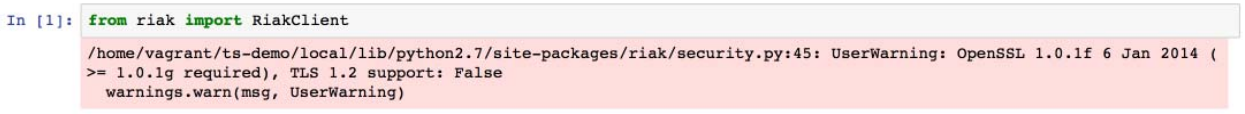
Click the **run** button below to run the Python code in that cell.



```
In [ ]: from riak import RiakClient

In [ ]: c=RiakClient()
        c.ping()
```

Once the code is run, the results will be displayed under that cell.

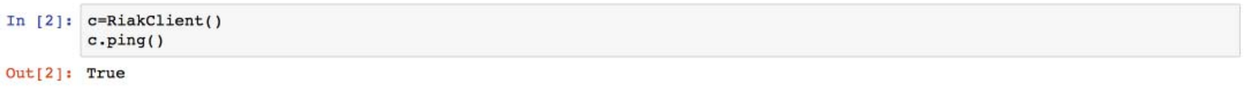
9. Import the RiakClient library for the Riak Python client.



```
In [1]: from riak import RiakClient
        /home/vagrant/ts-demo/local/lib/python2.7/site-packages/riak/security.py:45: UserWarning: OpenSSL 1.0.1f 6 Jan 2014 (
        >= 1.0.1g required), TLS 1.2 support: False
          warnings.warn(msg, UserWarning)
```

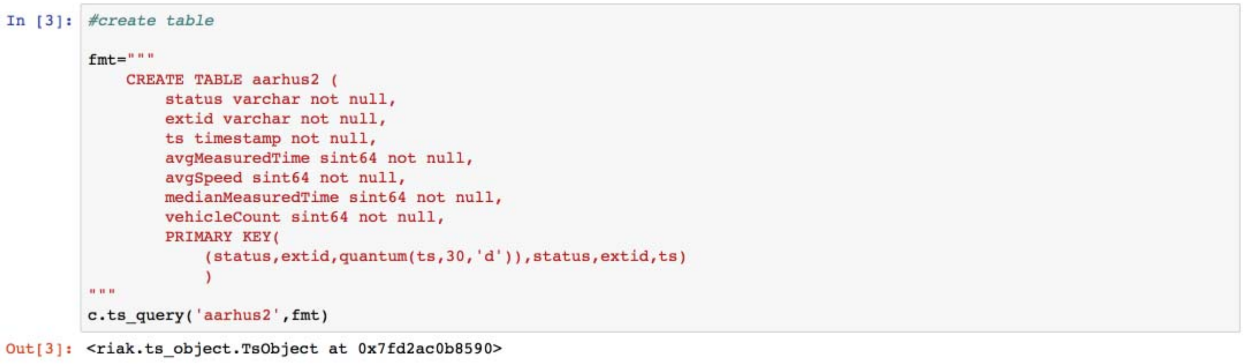
The OpenSSL warning happens whether you use the Jupyter notebooks or run this from the command line or Python script, and only occurs the first time you run this.

10. Create a Python Riak Client and ping Riak TS to verify you have a valid connection to the database, which you do by getting a 'True' response. A failure response will throw an exception.



```
In [2]: c=RiakClient()
        c.ping()
Out[2]: True
```

11. Create a table, aarhus2 using SQL, using the data from the CSV file



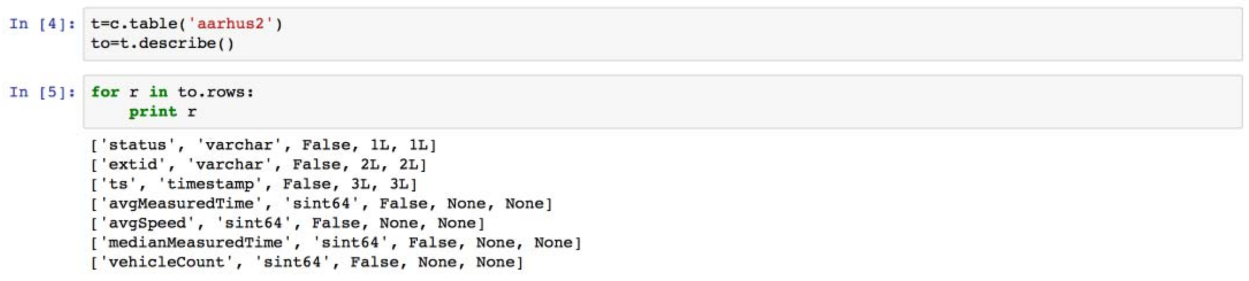
```
In [3]: #create table

        fmt="""
            CREATE TABLE aarhus2 (
                status varchar not null,
                extid varchar not null,
                ts timestamp not null,
                avgMeasuredTime sint64 not null,
                avgSpeed sint64 not null,
                medianMeasuredTime sint64 not null,
                vehicleCount sint64 not null,
                PRIMARY KEY(
                    (status,extid,quantum(ts,30,'d')),status,extid,ts)
                    )
        """
        c.ts_query('aarhus2',fmt)
Out[3]: <riak.ts_object.TsObject at 0x7fd2ac0b8590>
```

If you run this command a second time, you will see the message: **Failed to create table aarhus2: already_active** because the table has already been created.

The **primary key** can be just the time stamp or it can also include one or more columns to quantify your data in a manner that best aligns with how you want to use your data.

Submit the SQL code to Riak TS using the Riak Python client, and get a Python typed reference pointer to the table, which is the result of this command printed under the cell containing the code.

12. Do a SQL Describe on the aarhus2 table and then print the details of the table in the next cell.



```
In [4]: t=c.table('aarhus2')
        to=t.describe()

In [5]: for r in to.rows:
            print r

        ['status', 'varchar', False, 1L, 1L]
        ['extid', 'varchar', False, 2L, 2L]
        ['ts', 'timestamp', False, 3L, 3L]
        ['avgMeasuredTime', 'sint64', False, None, None]
        ['avgSpeed', 'sint64', False, None, None]
        ['medianMeasuredTime', 'sint64', False, None, None]
        ['vehicleCount', 'sint64', False, None, None]
```

Load Sensor Data Into Table aarhus2

Once the table has been created, you can start loading the sensor data into it. If you are collecting data in real time, you would be able to enter the data into the table as your application receives it.

There are over 1M rows of sensor data in the csv file. You will load the data using a Python script from the command line, since the script will print status information after each of our small batch writes.

13. ssh into riak-ts1, from a command line on your host machine.

    ```
    vagrant ssh riak-ts1
    ```

14. Activate the Python virtual environment (virtualenv), which is set up with the Riak Python client, pandas, and matplotlib.

    Using a virtualenv with Python is a good idea if you want to ensure that multiple projects on the same server don't compete for resources or use different versions of libraries or code. When you wish to use the virtualenv, **activate** it. When you are done with the virtualenv, **deactivate** it.

    ```
    cd ts-demo/
    source bin/activate
    ```

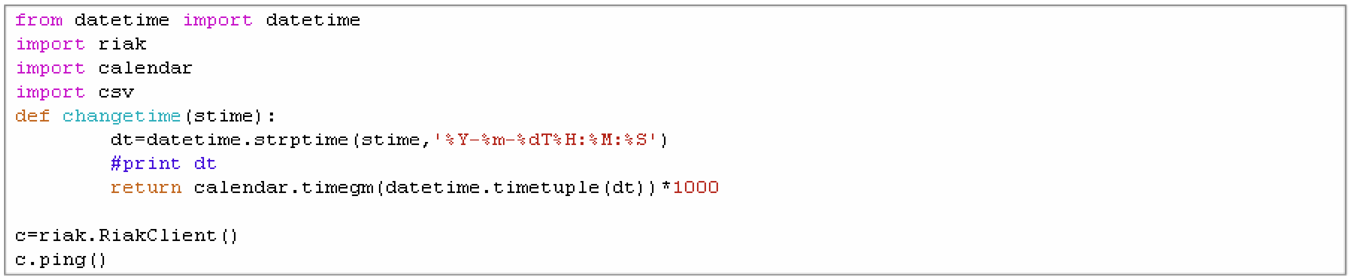
    When you are in the virtualenv, your prompt will prepend `(ts-demo)`, the name of your virtual environment to the prompt. So your prompt will now look like:

    ```
    (ts-demo) vagrant@riak-ts1:~/ts-demo$
    ```

15. Load the sensor data by executing the load-data-v2.py script. More information on timestamps and writing data to Riak TS can be found here.

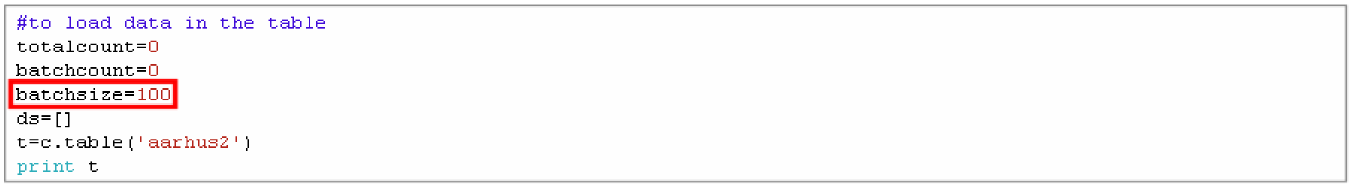
    ```
    python load-data-v2.py
    ```

    This piece of code imports Riak's Python Client, and defines a **changetime** function to convert data time in human readable format to UTC Epoch time, which is the format Riak TS expects for timestamp information. Ping the database to verify a valid connection.



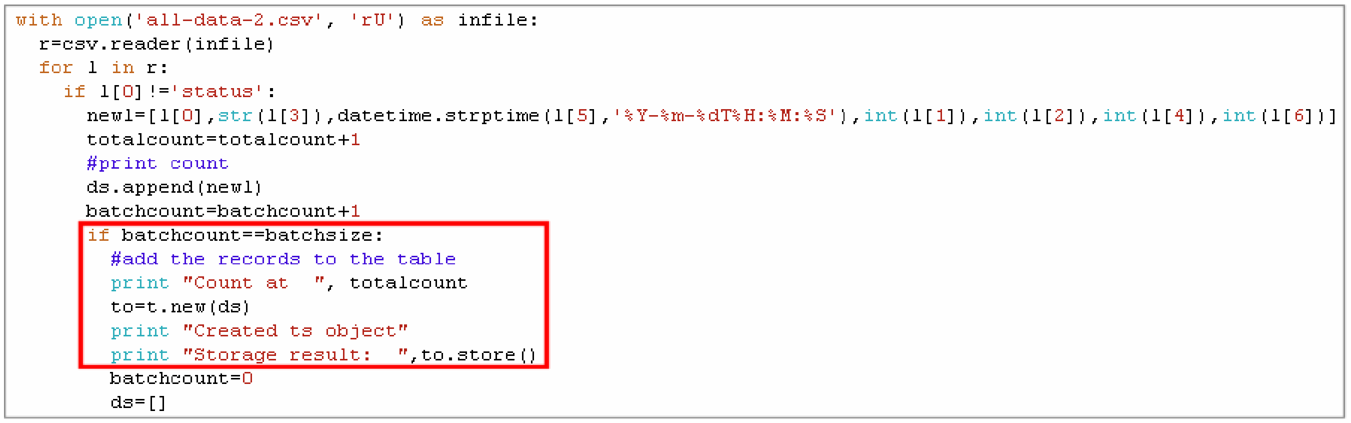
```
from datetime import datetime
import riak
import calendar
import csv
def changetime(stime):
        dt=datetime.strptime(stime,'%Y-%m-%dT%H:%M:%S')
        #print dt
        return calendar.timegm(datetime.timetuple(dt))*1000

c=riak.RiakClient()
c.ping()
```

We will write data in batches of 100 sensor readings



```
#to load data in the table
totalcount=0
batchcount=0
batchsize=100
ds=[]
t=c.table('aarhus2')
print t
```

Write batches of 5000 records to Riak TS, print result of write – successful write returns ‘True’, failure will throw an exception. The whole process should take a few minutes.



```
with open('all-data-2.csv', 'rU') as infile:
  r=csv.reader(infile)
  for l in r:
    if l[0]!='status':
      newl=[l[0],str(l[3]),datetime.strptime(l[5],'%Y-%m-%dT%H:%M:%S'),int(l[1]),int(l[2]),int(l[4]),int(l[6])]
      totalcount=totalcount+1
      #print count
      ds.append(newl)
      batchcount=batchcount+1
      if batchcount==batchsize:
        #add the records to the table
        print "Count at  ", totalcount
        to=t.new(ds)
        print "Created ts object"
        print "Storage result:  ",to.store()
        batchcount=0
        ds=[]
```

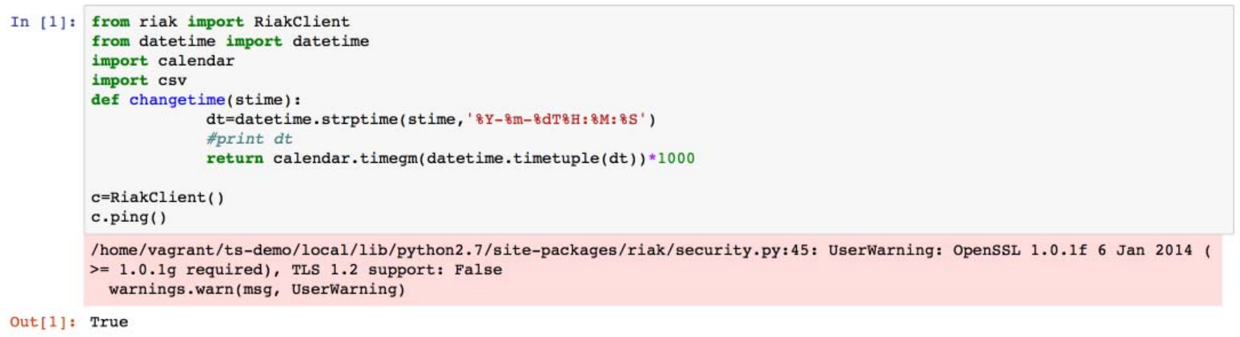
16. When you’re done with the virtualenv, it’s a good idea to deactivate it.

    `deactivate`

Query Your Data

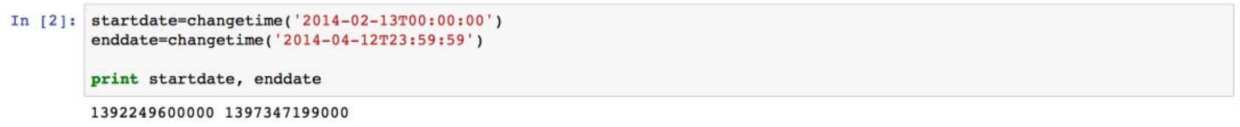
Now that you’ve loaded the data, let’s take a look at our data and answer some questions. As you did with the **Create table aarhus2** Notebook, go to the **Home** tab of Jupyter and double click on the **Query aarhus2** notebook.

17. This piece of code imports Riak’s Python Client, defines the same **changetime** function you used to load the data. Ping the database to verify a valid connection.



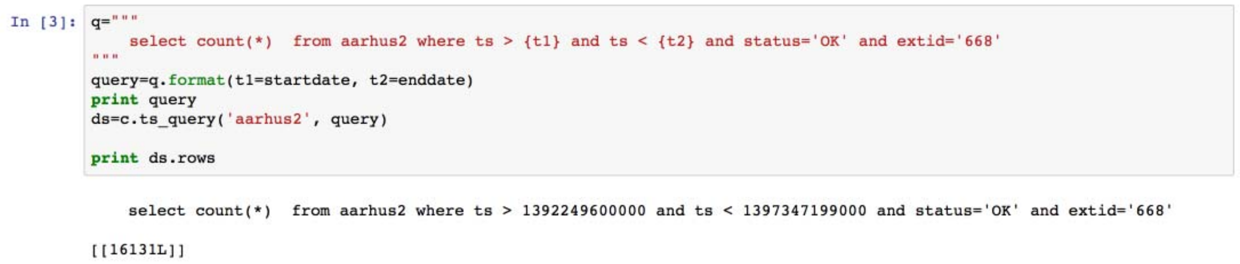
```
In [1]: from riak import RiakClient
        from datetime import datetime
        import calendar
        import csv
        def changetime(stime):
                    dt=datetime.strptime(stime,'%Y-%m-%dT%H:%M:%S')
                    #print dt
                    return calendar.timegm(datetime.timetuple(dt))*1000

        c=RiakClient()
        c.ping()

        /home/vagrant/ts-demo/local/lib/python2.7/site-packages/riak/security.py:45: UserWarning: OpenSSL 1.0.1f 6 Jan 2014 (
        >= 1.0.1g required), TLS 1.2 support: False
          warnings.warn(msg, UserWarning)

Out[1]: True
```

18. You will enter the start and end dates of the data you have collected from the sensors, which is February 13, 2014 through April 12, 2014. Convert these dates to UTC Epoch time.



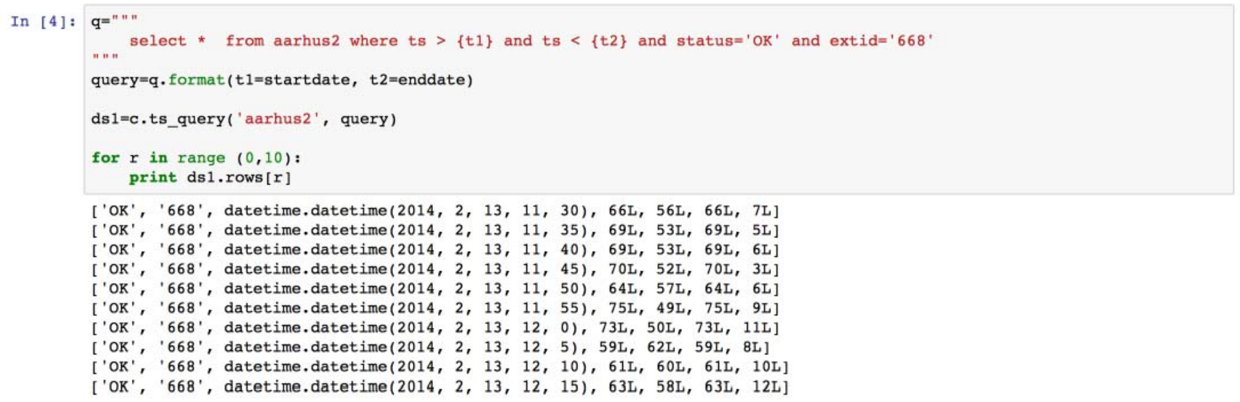
```
In [2]: startdate=changetime('2014-02-13T00:00:00')
        enddate=changetime('2014-04-12T23:59:59')

        print startdate, enddate

        1392249600000 1397347199000
```

19. Your first query will use the SQL count function to count the number of data points you have for status = ‘OK’ and ExtID = ‘668’.



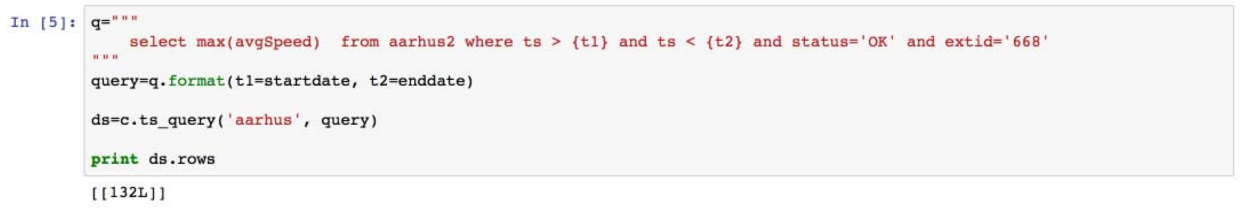
```
In [3]: q="""
            select count(*)  from aarhus2 where ts > {t1} and ts < {t2} and status='OK' and extid='668'
        """
        query=q.format(t1=startdate, t2=enddate)
        print query
        ds=c.ts_query('aarhus2', query)

        print ds.rows

            select count(*)  from aarhus2 where ts > 1392249600000 and ts < 1397347199000 and status='OK' and extid='668'

        [[16131L]]
```

20. Next you can view the data that was collected. Since there are over 16,000 records that fits this criteria, this will print the first 10 rows of data.



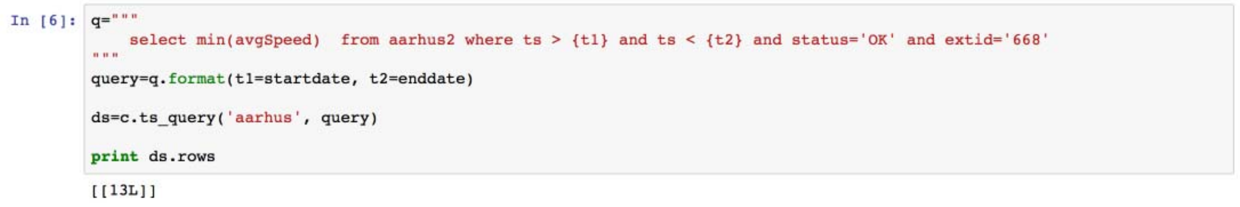
```
In [4]: q="""
            select *  from aarhus2 where ts > {t1} and ts < {t2} and status='OK' and extid='668'
        """
        query=q.format(t1=startdate, t2=enddate)

        ds1=c.ts_query('aarhus2', query)

        for r in range (0,10):
            print ds1.rows[r]

        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 30), 66L, 56L, 66L, 7L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 35), 69L, 53L, 69L, 5L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 40), 69L, 53L, 69L, 6L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 45), 70L, 52L, 70L, 3L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 50), 64L, 57L, 64L, 6L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 11, 55), 75L, 49L, 75L, 9L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 12, 0), 73L, 50L, 73L, 11L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 12, 5), 59L, 62L, 59L, 8L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 12, 10), 61L, 60L, 61L, 10L]
        ['OK', '668', datetime.datetime(2014, 2, 13, 12, 15), 63L, 58L, 63L, 12L]
```

21. Now you will find the maximum average speed.



```
In [5]: q="""
            select max(avgSpeed)  from aarhus2 where ts > {t1} and ts < {t2} and status='OK' and extid='668'
        """
        query=q.format(t1=startdate, t2=enddate)

        ds=c.ts_query('aarhus', query)

        print ds.rows
        [[132L]]
```

22. Find the minimum average speed.



```
In [6]: q="""
            select min(avgSpeed)  from aarhus2 where ts > {t1} and ts < {t2} and status='OK' and extid='668'
        """
        query=q.format(t1=startdate, t2=enddate)

        ds=c.ts_query('aarhus', query)

        print ds.rows
        [[13L]]
```

You can see how easy it is to answer questions of your data using SQL commands as you collect the information. If you were collecting data in real time, you saw how we can view all the data collected, record how many data points were collected at the time we are asking the question, and use aggregate functions to find maximum and minimum numbers that would be useful in urban planning for the Smart City.

Riak TS has other aggregate functions and we are continuing to add new functions and features with each release. You'll be able to do rolling upgrades to take advantage of the offerings of each release, so you won't have to plan downtime to do upgrades.

Riak Shell

Riak TS also provides a command line interface for running SQL statements called Riak Shell. You'll run the same SQL statements you just saw, and see that we get the same results. The SQL statements are in a text file **sql-notes-v2.txt**. The **last** SQL statement in the file is one you have **not yet** executed in Jupyter. It is the second to last SQL statement in the notebook.

23. At the command line, type (no password is required):

```
sudo riak-shell
```

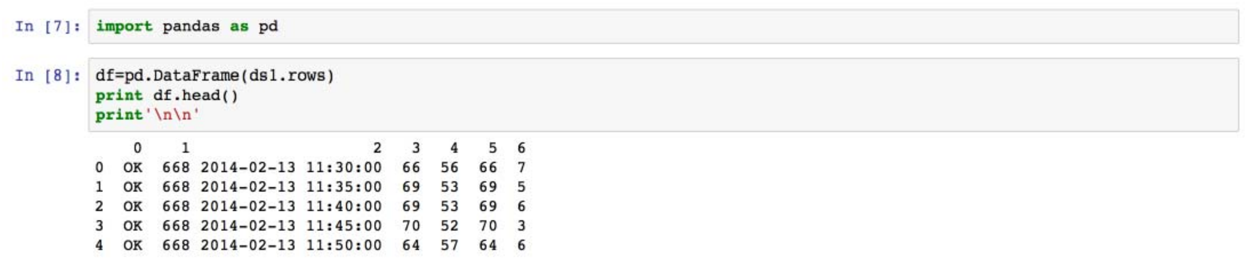
24. Type (or copy and paste) the SQL statements at the command line, and see that they give the same results as you saw in the Jupyter notebook.
25. When you are finished with the riak-shell, type (including the semi-colon at the end, because this is an Erlang environment):

```
quit;
```

Pandas DataFrame

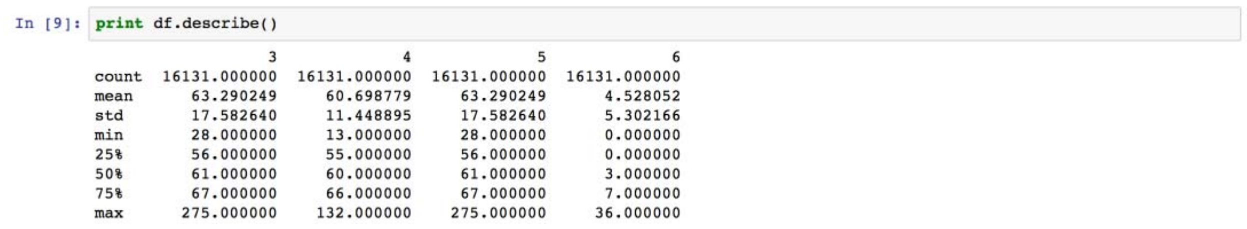
Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

26. In these cells we import pandas and print the first 5 rows of the data frame.



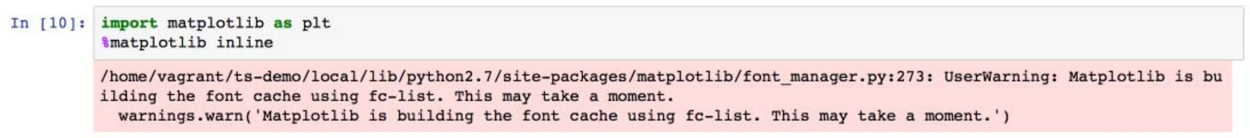
```
In [7]: import pandas as pd

In [8]: df=pd.DataFrame(ds1.rows)
        print df.head()
        print'\n\n'

           0    1                    2   3   4   5  6
        0  OK  668 2014-02-13 11:30:00  66  56  66  7
        1  OK  668 2014-02-13 11:35:00  69  53  69  5
        2  OK  668 2014-02-13 11:40:00  69  53  69  6
        3  OK  668 2014-02-13 11:45:00  70  52  70  3
        4  OK  668 2014-02-13 11:50:00  64  57  64  6
```

27. This shows some of pandas’ other analytical abilities that can be applied to the data you’ve gathered with Riak TS.



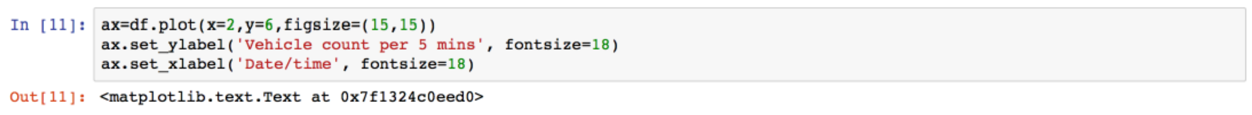
```
In [9]: print df.describe()

                          3              4              5              6
        count  16131.000000   16131.000000   16131.000000   16131.000000
        mean      63.290249      60.698779      63.290249       4.528052
        std       17.582640      11.448895      17.582640       5.302166
        min       28.000000      13.000000      28.000000       0.000000
        25%       56.000000      55.000000      56.000000       0.000000
        50%       61.000000      60.000000      61.000000       3.000000
        75%       67.000000      66.000000      67.000000       7.000000
        max      275.000000     132.000000     275.000000      36.000000
```

28. Now you will see how matplotlib can be used to graph your information to provide another way to analyze your data. The warning you see is only displayed on the first run to indicate it may be slow on a system with many fonts installed. It does not adversely affect our environment.

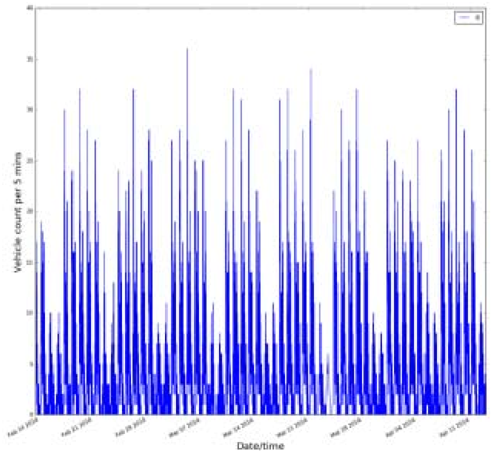


```
In [10]: import matplotlib as plt
         %matplotlib inline

         /home/vagrant/ts-demo/local/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarning: Matplotlib is bu
         ilding the font cache using fc-list. This may take a moment.
           warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')
```

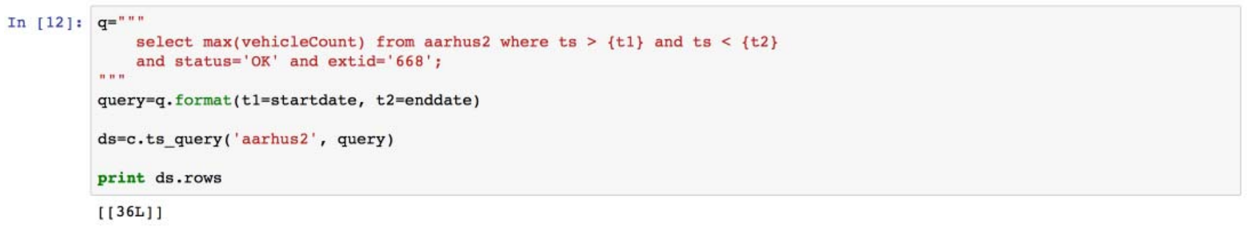
29. Set the parameters for our graph. Vehicle count per 5 minute interval along the y axis, and Date/time along the x axis. (picture of graph is reduced in size).



```
In [11]: ax=df.plot(x=2,y=6,figsize=(15,15))
         ax.set_ylabel('Vehicle count per 5 mins', fontsize=18)
         ax.set_xlabel('Date/time', fontsize=18)

Out[11]: <matplotlib.text.Text at 0x7f1324c0eed0>
```

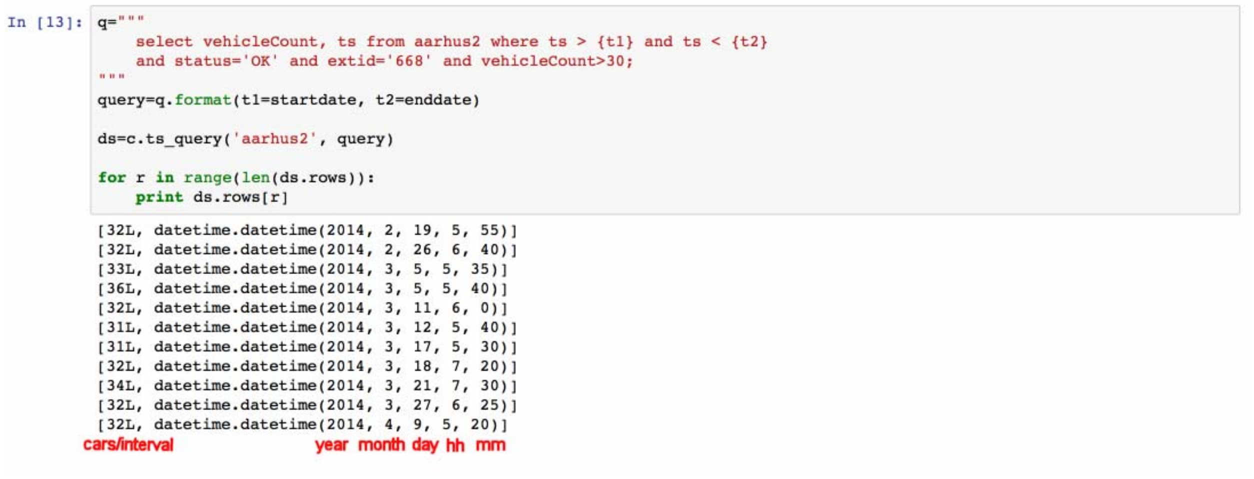


30. Now you will determine the maximum number of cars in a 5 minute interval. In a real world situation this might be important information for design plans for easing traffic congestion.



```
In [12]: q="""
             select max(vehicleCount) from aarhus2 where ts > {t1} and ts < {t2}
             and status='OK' and extid='668';
         """
         query=q.format(t1=startdate, t2=enddate)

         ds=c.ts_query('aarhus2', query)

         print ds.rows

         [[36L]]
```

31. Now you will determine the times that the number of cars exceeds 30 per interval.



```
In [13]: q="""
             select vehicleCount, ts from aarhus2 where ts > {t1} and ts < {t2}
             and status='OK' and extid='668' and vehicleCount>30;
         """
         query=q.format(t1=startdate, t2=enddate)

         ds=c.ts_query('aarhus2', query)

         for r in range(len(ds.rows)):
             print ds.rows[r]

         [32L, datetime.datetime(2014, 2, 19, 5, 55)]
         [32L, datetime.datetime(2014, 2, 26, 6, 40)]
         [33L, datetime.datetime(2014, 3, 5, 5, 35)]
         [36L, datetime.datetime(2014, 3, 5, 5, 40)]
         [32L, datetime.datetime(2014, 3, 11, 6, 0)]
         [31L, datetime.datetime(2014, 3, 12, 5, 40)]
         [31L, datetime.datetime(2014, 3, 17, 5, 30)]
         [32L, datetime.datetime(2014, 3, 18, 7, 20)]
         [34L, datetime.datetime(2014, 3, 21, 7, 30)]
         [32L, datetime.datetime(2014, 3, 27, 6, 25)]
         [32L, datetime.datetime(2014, 4, 9, 5, 20)]
        cars/interval              year  month day hh  mm
```

You can see that the number of cars in the left column and the hour and minutes on the right. The time when the traffic is heaviest (over 30 cars per interval) happens before 7:30am.

As you can see, Riak TS makes it easy to ask questions of your data using SQL statements and aggregations. We can easily integrate with other analysis tools such as pandas and matplotlib to give you more ways to get value from your data.

Riak TS is built on the same foundation as Riak KV, known in the industry for its high availability, resilience, fault tolerance, horizontal scalability using commodity hardware, and simplicity of operational management.

Sandbox Management

If you want to suspend the Sandbox environment, you can do so by using the command:

```
vagrant suspend
```

This will save the VM state of the 3 Riak TS nodes and suspend execution.

When you want to use the Sandbox again, resume the session by using the command

```
vagrant up
```

The Sandbox will be resumed from where you left off.

If you want to remove the Sandbox from your environment:

```
vagrant destroy [-f]
```

This command stops the running machine Vagrant is managing and destroys all resources that were created during the machine creation process. After running this command, your computer should be left at a clean state, as if you never created the guest machine in the first place.

The optional `[-f]` will destroy the 3 Riak TS servers without requiring a command line confirmation for each server.

If you wish to rebuild the Sandbox at a later date, you can use the `vagrant up` command to rebuild from scratch.

You can also check the status of the Virtual Box resources by launching the Virtual Box graphical interface:

```
VirtualBox
```

Additional Information

This section contains pointers to useful background information on Riak TS not directly covered by Getting Started. This section provides links to information such as how to stand up your own instance or instances of Riak TS.

If you'd like to customize your sandbox environment, see here for examples. **Be very careful you don't go below the minimal values set (2 GB RAM, 2 CPUs, 3 nodes of Riak TS).**

https://github.com/basho-labs/riak-demos

Information on latest version of Riak TS
http://docs.basho.com/riakts/latest/

Release Notes
http://docs.basho.com/riakts/latest/releasenotes/

Planning your Riak TS Table
http://docs.basho.com/riakts/latest/using/planning/

Writing Data to Riak TS
http://docs.basho.com/riakts/latest/using/writingdata/

Querying Data in Riak TS
http://docs.basho.com/riakts/latest/using/querying/

Configuring Riak TS
http://docs.basho.com/riakts/latest/using/configuring/

Aggregation Functions
http://docs.basho.com/riakts/latest/using/aggregate-functions/

Arithmetic Functions
http://docs.basho.com/riakts/latest/using/arithmetic-operations/

Riak Shell
http://docs.basho.com/riakts/latest/using/riakshell/

Supported Riak Clients
http://docs.basho.com/riak/ts/latest/developing/

SQL for Riak TS
http://docs.basho.com/riakts/latest/learn-about/sqlriakts/

Standing up your own Riak TS Cluster
https://github.com/basho-labs/riak-demos/wiki/Provision-a-cluster-outside-of-vagrant

This will give you instructions and code to stand up your own Riak TS cluster. The Sandbox takes the additional steps of creating the **ts-demo** directory, and adding the files for the demos made available in the demo environment.

If you wish to have the demo files, they are in the **ts-demo** directory in the **tar** files. Just scp them to your own environment.

The Riak community is an incredible collection of developers. You can connect with fellow members at:

http://basho.com/community/