Final Project

Amazon Kinesis Firehose

Gallagher, Katherine

cscie90 Cloud Computing

Harvard University Extension School

Prof. Zoran B. Djordjević

Topic: Amazon Kinesis Firehose
Project Summary

Project Description: I will be using Amazon Kinesis Firehose to stream filtered tweets in (near) real-time to Amazon S3 and Amazon Redshift.

Problem: Our increasingly connected world generates massive amounts of data. Amazon Kinesis originally sought to provide an integrated solution to data collection and analysis by providing a platform on which to build custom applications that capture, process, and store streaming data. While this proved useful for technologically literate users, Amazon realized that others would find a simple data delivery stream preferable. Amazon Kinesis Firehose, "the easiest way to load streaming data into AWS", was the result of this realization.

Description: Introduced in 2013, Amazon Kinesis Firehose is a pay-as-you-go autoscaling delivery stream capable of loading gigabytes of streaming data per second from hundreds of thousands of sources into Amazon S3 or Amazon Redshift in near real-time (within 60 seconds of sending). While Firehose allows considerable flexibility in detail specification, allowing users to set batch size, interval, compression, encryption, etc., it is a fully-managed service, and self-administers the resources required for data loading.

Benefits: The primary benefit to Firehose is ease of use; a delivery stream can be up and running with a few clicks in the AWS Kinesis console and requires no further administration. Rather than requiring a minimum spend, users pay per GB of data ingested, so Firehose is equally appealing to small and large data consumers. As the transport data unit is opaque to the delivery stream, Firehose does not discriminate between data formats.

Drawbacks: While the S3 integration appears seamless, I found Redshift less flexible to work with programmatically. That said, this is more a complaint with Redshift than Firehose, and can be easily circumvented by setting options in the AWS console or deploying from SQL.

Data Set: Filtered tweets captured from the Twitter Public Stream (Resource URL: https://stream.twitter.com/1.1/statuses/filter.json)

Operating System: Mac OS X Yosemite - Version 10.10.5

Software:
- SQL Workbench/J
- Eclipse Java EE IDE for Web Developers - Version Mars.1 Release (4.5.1)

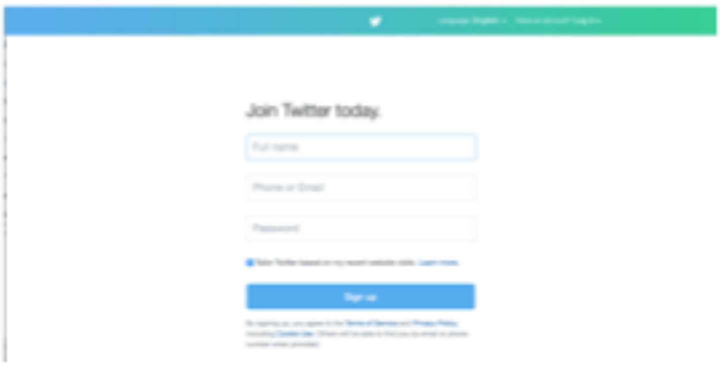
Overview of steps:
- Set up Twitter account and Twitter application to obtain credentials for Twitter API calls
- Create S3 bucket, Redshift cluster, and Redshift table to hold incoming data
- Create a Firehose delivery stream connected to S3 and Redshift
- Connect to Twitter Public Stream and send tweets via Firehose

Links to YouTube videos:
- Short: https://youtu.be/hBHbgOtiy18
- Long: https://youtu.be/WyO30uaz7Fw

Amazon Kinesis Firehose

To put Firehose's delivery capabilities to the test, we want to send through an unlimited amount of streaming data. With approximately 500 million tweets per day (and growing!)[1] and a user-friendly API, Twitter is an excellent candidate for generating our test data.

We first need to set up a Twitter account.



Join Twitter today.

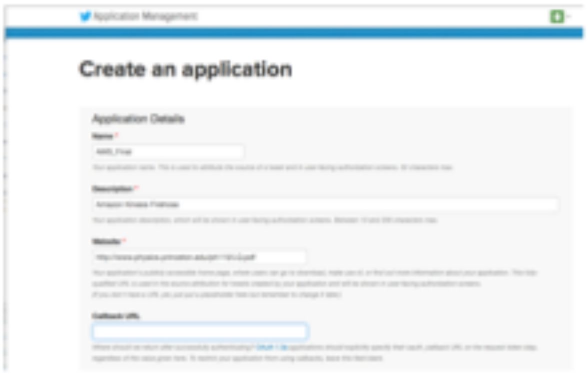
Full name

Phone or Email

Password

Sign up

Then, we can create a Twitter application, which provides credentials needed to make API calls.



Application Management

Create an application

Application Details

Name *

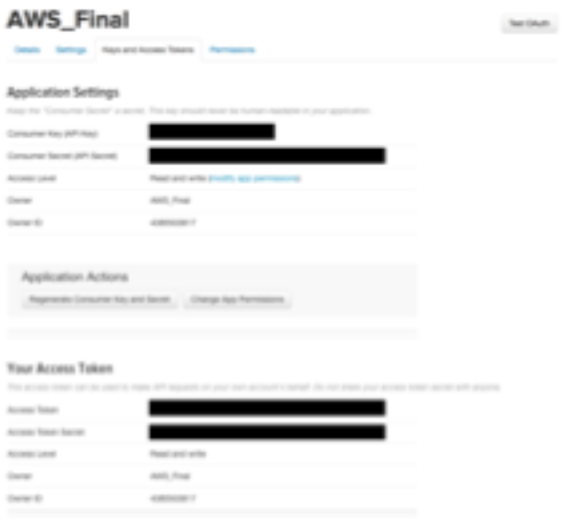
AWS_Final

Description *

Amazon Kinesis Firehose

Website *

Callback URL



AWS_Final

Test OAuth

Details | Settings | Keys and Access Tokens | Permissions

Application Settings

Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read and write (modify app permissions)
Owner	AWS_Final
Owner ID	[illegible]

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

Access Token	[redacted]
Access Token Secret	[redacted]
Access Level	Read and write
Owner	AWS_Final
Owner ID	[illegible]

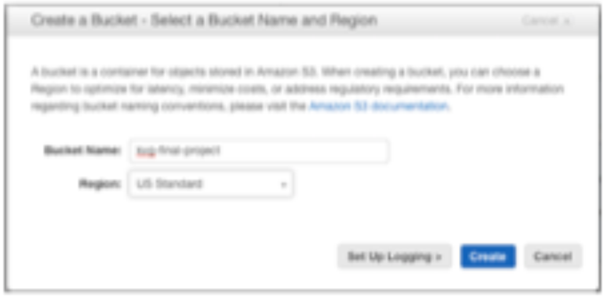
The Resource URL for Twitter's public streaming API can be found in the Twitter dev docs:

```
https://stream.twitter.com/1.1/statuses/filter.json
```

[1] "Twitter Usage Statistics." Internet Live Stats. Web. 10 Dec. 2015.

Now that we have what we need to access Twitter's public stream, we need repositories to hold the incoming data. Our data will first be stored in an S3 bucket, then copied to a Redshift table.

Set up an S3 bucket and Redshift cluster:



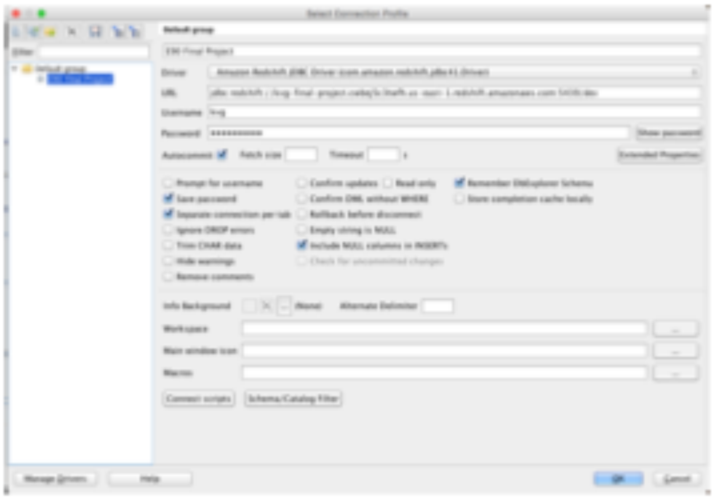
Create a Bucket - Select a Bucket Name and Region

A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the Amazon S3 documentation.

Bucket Name: kvg-final-project

Region: US Standard

Connect to default Redshift database "dev" using SQL Workbench/J, a cross-platform SQL query tool. This requires downloading the Amazon Redshift JDBC Driver from AWS as a .jar file, and adding the driver and Redshift cluster's JDBC URL to the SQL Connection Profile:



Select Connection Profile

Default group

E90 Final Project

Driver: Amazon Redshift JDBC Driver (com.amazon.redshift.jdbc41.Driver)

URL: jdbc:redshift://kvg-final-project.[illegible].us-east-1.redshift.amazonaws.com:5439/dev

Username: kvg

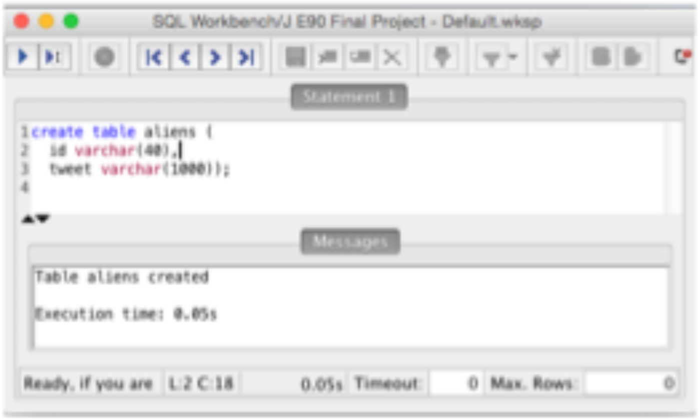
Password: **********

Autocommit ✓

✓ Save password
✓ Separate connection per tab
✓ Include NULL columns in INSERTs
✓ Remember DbExplorer Schema

Now connected to our Redshift database, we create a table for data retrieved from S3.

Table "aliens" has two columns, "id" and "tweet". Each S3 object will be a text file of a string containing a tweet ID and the text of the tweet. When the objects are copied to Redshift, the database splits the string at a programmed delimiter "|", sending the tweet ID to column "id" and the text to column "tweet".



SQL Workbench/J E90 Final Project - Default.wksp

Statement 1

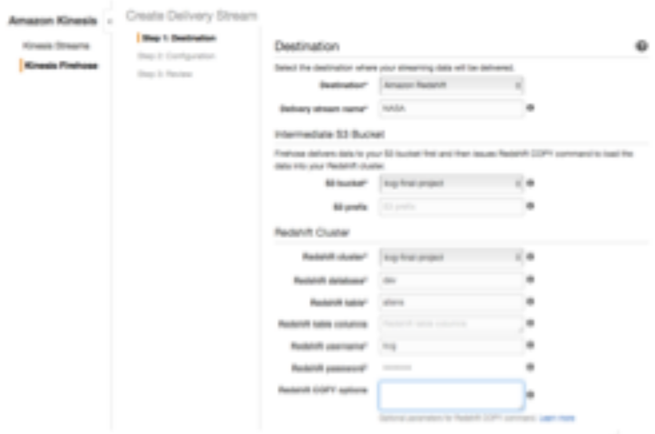
```
create table aliens (
 id varchar(40),
 tweet varchar(1000));
```

Messages

```
Table aliens created

Execution time: 0.05s
```

Ready, if you are | L:2 C:18 | 0.05s | Timeout: 0 | Max. Rows: 0

While we are eventually going to create a Firehose delivery stream programmatically, it is worthwhile first going through the AWS console to see how user-friendly it is to set up:


Create Delivery Stream

Destination

[illegible]

Destination*	Amazon Redshift
Delivery stream name*	NASA

Intermediate S3 Bucket

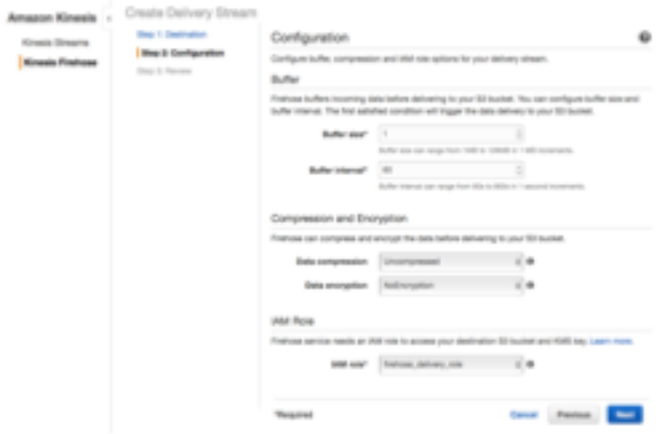
[illegible]

S3 bucket*	kvg final project
S3 prefix	

Redshift Cluster

Redshift cluster*	kvg final project
Redshift database*	dev
Redshift table*	aliens
Redshift table columns	
Redshift username*	kvg
Redshift password*	
Redshift COPY options	

Destination: “Amazon S3” will stop stream delivery at the S3 bucket, while “Amazon Redshift” will first deliver to the S3 bucket, then copy the S3 objects to the specified Redshift table.


Create Delivery Stream

Configuration

[illegible]

Buffer

[illegible]

Buffer size*	1
Buffer interval*	60

Compression and Encryption

[illegible]

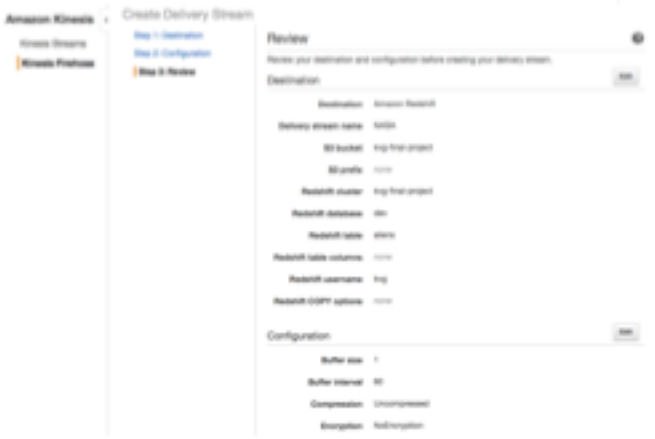
Data compression	Uncompressed
Data encryption	NoEncryption

IAM Role

[illegible]

IAM role*	firehose_delivery_role

Configuration: We want low latency for near real-time streaming, so a small buffer size and buffer interval is ideal. We do not need compression or encryption, but we do need an IAM role that permits Firehose access to our repositories.


Create Delivery Stream

Review

[illegible]

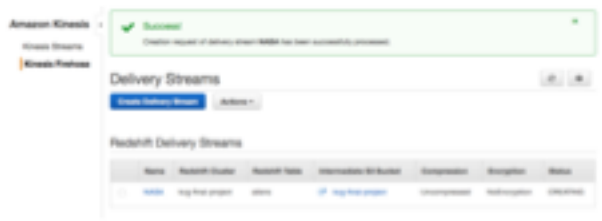
Destination

Destination	Amazon Redshift
Delivery stream name	NASA
S3 bucket	kvg final project
S3 prefix	none
Redshift cluster	kvg final project
Redshift database	dev
Redshift table	aliens
Redshift table columns	none
Redshift username	kvg
Redshift COPY options	none

Configuration

Buffer size	1
Buffer interval	60
Compression	Uncompressed
Encryption	NoEncryption

Review settings, click “create”…


Success!

Creation request of delivery stream NASA has been successfully processed.

Delivery Streams

Redshift Delivery Streams

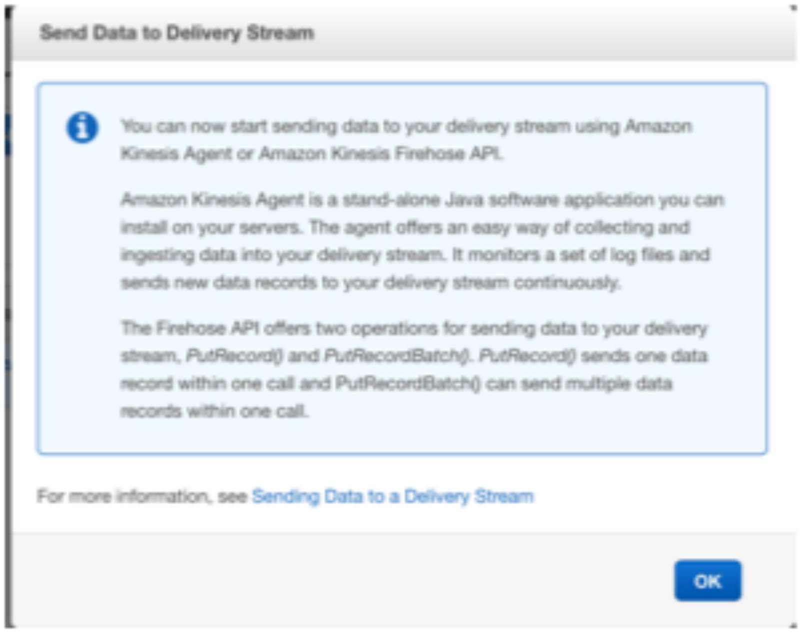
Name	Redshift Cluster	Redshift Table	Intermediate S3 Bucket	Compression	Encryption	Status
NASA	kvg final project	aliens	kvg final project	Uncompressed	NoEncryption	CREATING

…and our delivery stream has been created.

If we now go to send data to our delivery stream, we are met with a message telling us that we

If we now go to send data to our delivery stream, we are met with a message telling us that we have two options: Amazon Kinesis Agent (AWS Java app), or Amazon Kinesis Firehose API.

Before moving to our program, let's look at the Amazon Kinesis Agent.


Send Data to Delivery Stream

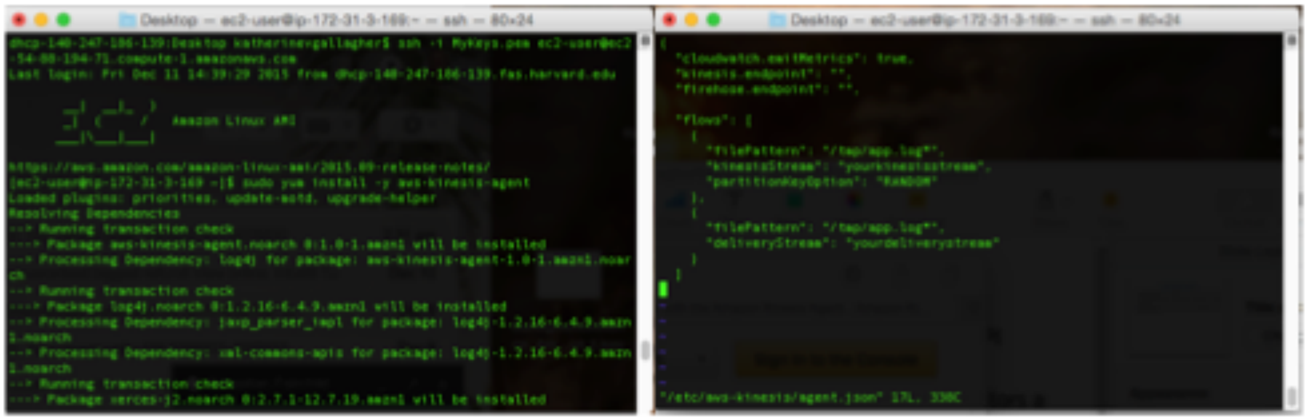
You can now start sending data to your delivery stream using Amazon Kinesis Agent or Amazon Kinesis Firehose API.

Amazon Kinesis Agent is a stand-alone Java software application you can install on your servers. The agent offers an easy way of collecting and ingesting data into your delivery stream. It monitors a set of log files and sends new data records to your delivery stream continuously.

The Firehose API offers two operations for sending data to your delivery stream, *PutRecord()* and *PutRecordBatch()*. *PutRecord()* sends one data record within one call and PutRecordBatch() can send multiple data records within one call.

For more information, see Sending Data to a Delivery Stream

Amazon Kinesis Agent requires an active delivery stream for set up. It is easy to install on a Linux EC2 Instance or other Linux server. To configure, edit /etc/aws-kinesis/agent.json with the location of your files ("filePattern") and the name and endpoint of the delivery stream:


Desktop — ec2-user@ip-172-31-3-169:~ — ssh — 80×24
Desktop — ec2-user@ip-172-31-3-169:~ — ssh — 80×24

```
dhcp-140-247-186-139:Desktop katherinevgallagher$ ssh -i MyKeys.pem ec2-user@ec2
-54-88-194-71.compute-1.amazonaws.com
Last login: Fri Dec 11 14:39:29 2015 from dhcp-140-247-186-139.fas.harvard.edu

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
[ec2-user@ip-172-31-3-169 ~]$ sudo yum install -y aws-kinesis-agent
Loaded plugins: priorities, update-motd, upgrade-helper
Resolving Dependencies
--> Running transaction check
---> Package aws-kinesis-agent.noarch 0:1.0-1.amzn1 will be installed
--> Processing Dependency: log4j for package: aws-kinesis-agent-1.0-1.amzn1.noar
ch
--> Running transaction check
---> Package log4j.noarch 0:1.2.16-6.4.9.amzn1 will be installed
--> Processing Dependency: jaxp_parser_impl for package: log4j-1.2.16-6.4.9.amzn
1.noarch
--> Processing Dependency: xml-commons-apis for package: log4j-1.2.16-6.4.9.amzn
1.noarch
--> Running transaction check
---> Package xerces-j2.noarch 0:2.7.1-12.7.19.amzn1 will be installed
```

```
{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "",
  "firehose.endpoint": "",

  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "yourkinesisstream",
      "partitionKeyOption": "RANDOM"
    },
    {
      "filePattern": "/tmp/app.log*",
      "deliveryStream": "yourdeliverystream"
    }
  ]
}
~
~
~
~
~
~
"/etc/aws-kinesis/agent.json" 17L, 330C
```

When deployed, the agent will continuously monitor the files at the specified location, and send the file data to Firehose. It will also emit CloudWatch metrics for user-friendly monitoring and trouble-shooting.

For this project, however, we are going to use the Amazon Kinesis Firehose API.

We have NASA.java ready to deploy in Eclipse:

```java
package twitterFirehose;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;

import org.scribe.builder.*;
import org.scribe.builder.api.*;
import org.scribe.model.*;
import org.scribe.oauth.*;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.kinesisfirehose.AmazonKinesisFirehose;
import com.amazonaws.services.kinesisfirehose.AmazonKinesisFirehoseClient;
import com.amazonaws.services.kinesisfirehose.model.BufferingHints;
import com.amazonaws.services.kinesisfirehose.model.CopyCommand;
import com.amazonaws.services.kinesisfirehose.model.CreateDeliveryStreamRequest;
import com.amazonaws.services.kinesisfirehose.model.DescribeDeliveryStreamRequest;
import com.amazonaws.services.kinesisfirehose.model.DescribeDeliveryStreamResult;
import com.amazonaws.services.kinesisfirehose.model.PutRecordRequest;
import com.amazonaws.services.kinesisfirehose.model.Record;
import com.amazonaws.services.kinesisfirehose.model.RedshiftDestinationConfiguration;
import com.amazonaws.services.kinesisfirehose.model.S3DestinationConfiguration;

public class NASA {

        private static final String firehoseName = "NASA";
        private static final String STREAM_URI = "https://stream.twitter.com/1.1/statuses/filter.json";
        private static final String key = "(Put Twitter Key here)";
        private static final String secret = "(Put Twitter Secret Key here";
        private static final String token = "(Put Twitter Token here)";
        private static final String tokenSecret = "(Put Twitter Token Secret here";
        private static final String searchTerm = "#NASA";

    public static void main(String[] args){

        // Set up AWS Credentials and Kinesis Firehose client

         AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location and in valid format.",
                    e);
        }

        AmazonKinesisFirehose firehose = new AmazonKinesisFirehoseClient(credentials);

        try {

        // Create new Kinesis Firehose deliveryStream

        System.out.println("Creating deliveryStream '" + firehoseName + "'\n");
        CreateDeliveryStreamRequest createDeliveryStreamRequest = new CreateDeliveryStreamRequest();
        createDeliveryStreamRequest.setDeliveryStreamName(firehoseName);
```

```java
        // Structure existing S3 bucket for data delivery

        S3DestinationConfiguration s3DestinationConfiguration = new S3DestinationConfiguration();

        String s3ARN = "arn:aws:s3:::kvn-final-project";
        String s3ARN = "arn:aws:s3:::kvg-final-project";
                        s3DestinationConfiguration.setBucketARN(s3ARN);

                        String s3Prefix = "NASA";
                        s3DestinationConfiguration.setPrefix(s3Prefix);

        String iamRoleARN = "arn:aws:iam::127497063198:role/firehose_delivery_role";
        s3DestinationConfiguration.setRoleARN(iamRoleARN);

        Integer destinationSizeInMBs = 1;
        Integer destinationIntervalInSeconds = 60;

        BufferingHints bufferingHints = null;
        if (destinationSizeInMBs != null || destinationIntervalInSeconds != null) {
            bufferingHints = new BufferingHints();
            bufferingHints.setSizeInMBs(destinationSizeInMBs);
            bufferingHints.setIntervalInSeconds(destinationIntervalInSeconds);
        }
        s3DestinationConfiguration.setBufferingHints(bufferingHints);

        // Structure existing Redshift database to receive copy of S3 contents
        CopyCommand copyCommand = new CopyCommand();
        copyCommand.withDataTableName("aliens");

        RedshiftDestinationConfiguration redshiftDestinationConfiguration = new
RedshiftDestinationConfiguration();

        redshiftDestinationConfiguration.withClusterJDBCURL("jdbc:redshift://kvg-final-
project.cwbej5c3twfh.us-east-1.redshift.amazonaws.com:5439/dev")
           .withRoleARN(iamRoleARN)
           .withUsername("kvg")
           .withPassword("(Put Redshift password here)")
           .withCopyCommand(copyCommand)
           .withS3Configuration(s3DestinationConfiguration);

        createDeliveryStreamRequest.setRedshiftDestinationConfiguration(redshiftDestinationConfiguration);
        firehose.createDeliveryStream(createDeliveryStreamRequest);

        // Give deliveryStream time to create
        System.out.println("Sleeping for 30 seconds to ensure finalized deliveryStream activation...\n");
        Thread.sleep(30000);

            DescribeDeliveryStreamRequest describeDeliveryStreamRequest = new DescribeDeliveryStreamRequest();
            describeDeliveryStreamRequest.withDeliveryStreamName(firehoseName);
            DescribeDeliveryStreamResult describeDeliveryStreamResponse =
                    firehose.describeDeliveryStream(describeDeliveryStreamRequest);
            String status =
describeDeliveryStreamResponse.getDeliveryStreamDescription().getDeliveryStreamStatus();

        if (status.equals("CREATING")){
            System.out.println("DeliveryStream not yet active; sleeping for an additional 10 seconds...\n");
            Thread.sleep(10000);
        }

            System.out.println("Firehose '" + firehoseName + "' created!\n");

         } catch (AmazonClientException | InterruptedException ace) {

             System.out.println("Error Message: " + ace.getMessage() + "\n");
         }
```

```
        try{
          System.out.println("Starting Twitter public stream consumer for search term '" + searchTerm + "'\n");

            // Pass Twitter consumer key and secret to OAuth
            OAuthService service = new ServiceBuilder()
                    .provider(TwitterApi.class)
                    .provider(TwitterApi.class)
                    .apiKey(key)
                    .apiSecret(secret)
                    .build();

            // Set Twitter access token and token secret
            Token accessToken = new Token(token, tokenSecret);

            // Generate the request to search public stream
            System.out.println("Connecting to Twitter Public Stream...\n");
            OAuthRequest request = new OAuthRequest(Verb.POST, STREAM_URI);
            request.addHeader("version", "HTTP/1.1");
            request.addHeader("host", "stream.twitter.com");
            request.setConnectionKeepAlive(true);
            request.addHeader("user-agent", "Twitter Stream Reader");
            request.addBodyParameter("track", searchTerm);
            service.signRequest(accessToken, request);
            Response response = request.send();

            // Create a reader to read Twitter stream
            BufferedReader reader = new BufferedReader(new InputStreamReader(response.getStream()));

            String line;
            while ((line = reader.readLine()) != null) {

                if (line.length() > 0){

                        String [] parts = line.split(",", 5);
                        String ID = parts[1];
                        String text = parts[3];
                        String tweet = ID + "|" + text + "\n";

                        System.out.println("SENDING TO S3 BUCKET: " + tweet);

                        // cast tweets to byte array
                        byte [] tweetBytes = tweet.getBytes();

                        // cast byte array to ByteBuffer (AWS Request data format)
                        ByteBuffer tweetBB = ByteBuffer.wrap(tweetBytes);

                        PutRecordRequest putRecordRequest = new PutRecordRequest();
                        putRecordRequest.setDeliveryStreamName(firehoseName);

                        Record record = new Record();
                        record.setData(tweetBB);
                        putRecordRequest.setRecord(record);

                        firehose.putRecord(putRecordRequest);
                        }
                }
        } catch (IOException ioe){
            ioe.printStackTrace();
        }

    }
}
```

(Twitter-related code adapted for use from Jeff Kutcha's blog)
(AWS-related code adapted for use from Amazon Kinesis Firehose Documentation)

When run, this program will create a new Amazon Kinesis Firehose delivery stream…

```
 AmazonKinesisFirehose firehose = new AmazonKinesisFirehoseClient(credentials);

        try {

        // Create new Kinesis Firehose deliveryStream
        // Create new Kinesis Firehose deliveryStream

        System.out.println("Creating deliveryStream '" + firehoseName + "'\n");
        CreateDeliveryStreamRequest createDeliveryStreamRequest = new CreateDeliveryStreamRequest();
        createDeliveryStreamRequest.setDeliveryStreamName(firehoseName);
```

and set it to deliver to our S3 bucket:

```
        // Structure existing S3 bucket for data delivery

        S3DestinationConfiguration s3DestinationConfiguration = new S3DestinationConfiguration();

        String s3ARN = "arn:aws:s3:::kvg-final-project";
                        s3DestinationConfiguration.setBucketARN(s3ARN);

                        String s3Prefix = "NASA";
                        s3DestinationConfiguration.setPrefix(s3Prefix);

        String iamRoleARN = "arn:aws:iam::127497063198:role/firehose_delivery_role";
        s3DestinationConfiguration.setRoleARN(iamRoleARN);

        Integer destinationSizeInMBs = 1;
        Integer destinationIntervalInSeconds = 60;

        BufferingHints bufferingHints = null;
        if (destinationSizeInMBs != null || destinationIntervalInSeconds != null) {
            bufferingHints = new BufferingHints();
            bufferingHints.setSizeInMBs(destinationSizeInMBs);
            bufferingHints.setIntervalInSeconds(destinationIntervalInSeconds);
        }
        s3DestinationConfiguration.setBufferingHints(bufferingHints);
```

It will also set the S3 bucket objects to subsequently be copied to our Redshift table.

```
        // Structure existing Redshift database to receive copy of S3 contents

        CopyCommand copyCommand = new CopyCommand();
        copyCommand.withDataTableName("aliens");

        RedshiftDestinationConfiguration redshiftDestinationConfiguration = new
RedshiftDestinationConfiguration();

        redshiftDestinationConfiguration.withClusterJDBCURL("jdbc:redshift://kvg-final-
project.cwbej5c3twfh.us-east-1.redshift.amazonaws.com:5439/dev")
            .withRoleARN(iamRoleARN)
            .withUsername("kvg")
            .withPassword("(Put Redshift password here)")
            .withCopyCommand(copyCommand)
            .withS3Configuration(s3DestinationConfiguration);

        createDeliveryStreamRequest.setRedshiftDestinationConfiguration(redshiftDestinationConfiguration);
        firehose.createDeliveryStream(createDeliveryStreamRequest);
```

Twitter uses OAuth authentication. With the help of Scribe, an OAuth Java library, the program will connect to the Twitter Public Stream and search for tweets that include our search term.

```
try{
         System.out.println("Starting Twitter public stream consumer for search term '" + searchTerm + "'\n");

           // Pass Twitter consumer key and secret to OAuth
           OAuthService service = new ServiceBuilder()
OAuthService service = new ServiceBuilder()
        .provider(TwitterApi.class)
        .apiKey(key)
        .apiSecret(secret)
        .build();

// Set Twitter access token and token secret
Token accessToken = new Token(token, tokenSecret);

// Generate the request to search public stream
System.out.println("Connecting to Twitter Public Stream...\n");
OAuthRequest request = new OAuthRequest(Verb.POST, STREAM_URI);
request.addHeader("version", "HTTP/1.1");
request.addHeader("host", "stream.twitter.com");
request.setConnectionKeepAlive(true);
request.addHeader("user-agent", "Twitter Stream Reader");
request.addBodyParameter("track", searchTerm);
service.signRequest(accessToken, request);
Response response = request.send();
```

The tweets are returned as a long string of Twitter data, from which we create a new string that uses only the tweet ID and text body, separated by a delimiter "|".

```
// Create a reader to read Twitter stream
BufferedReader reader = new BufferedReader(new InputStreamReader(response.getStream()));

String line;
while ((line = reader.readLine()) != null) {

    if (line.length() > 0){
            String [] parts = line.split(",", 5);
            String ID = parts[1];
            String text = parts[3];
            String tweet = ID + "|" + text + "\n";

            System.out.println("SENDING TO S3 BUCKET: " + tweet);
```

Each string is cast to a byte array for easy conversion to a byte buffer, the required format from which to generate a Kinesis record, the unit of data transported by delivery streams.

```
            // cast tweets to byte array
            byte [] tweetBytes = tweet.getBytes();
            // cast byte array to ByteBuffer (AWS Request data format)
            ByteBuffer tweetBB = ByteBuffer.wrap(tweetBytes);

            PutRecordRequest putRecordRequest = new PutRecordRequest();
            putRecordRequest.setDeliveryStreamName(firehoseName);

            Record record = new Record();
            record.setData(tweetBB);
            putRecordRequest.setRecord(record);

            firehose.putRecord(putRecordRequest);
            }
```

As Kinesis does not interact with the data blob carried by a record, Firehose can easily transport any datatype without prejudice.

The Firehose delivery stream will first deliver the records to our S3 bucket, where they are saved as text files. The text files will then be copied into our Redshift table, which looks for the delimiter we added to split the string into our two columns, "id" and "tweet".

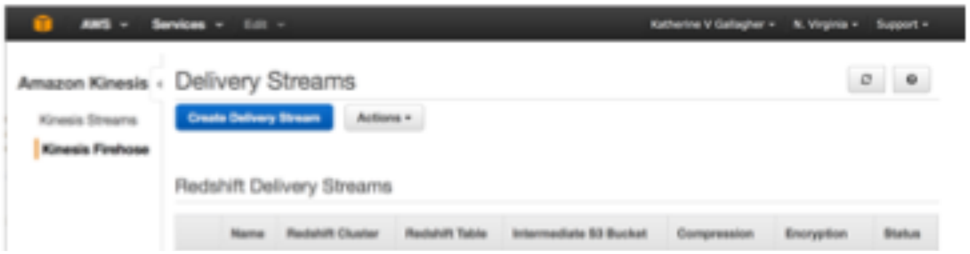
Output:

Output:

```
Creating deliveryStream 'NASA'

Sleeping for 30 seconds to ensure finalized deliveryStream activation...

DeliveryStream not yet active; sleeping for an additional 10 seconds...

Firehose 'NASA' created!

Starting Twitter public stream consumer for search term '#NASA'

Connecting to Twitter Public Stream...

SENDING TO S3 BUCKET: "id":674791746421657600|"text":"NASA\u304c\u767a
\u8868\u3057\u305f\u6e96\u60d1\u661f\u30b1\u30ec\u30b9\u306e\u5168\u4f53\u50cf
\uff01\u2b50\ufe0f\u3053\u306e\u9752\u767d\u304f\u5149\u3063\u3066\u3044\u308b
\u306e\u306f\u5869\u3089\u3057\u3044\u2026\u3093\u30fc\u3059\u3054\u3044\u8feb
\u529b\uff01\ud83d\ude32\n#NASA #Ceres #space https:\/\/t.co\/nBb2quak3K"

SENDING TO S3 BUCKET: "id":674791760938291201|"text":"Wednesday December 9
2015:10:25:50 PM EST we live on a #FlatEarth. #NASA &amp; others deceived us.
https:\/\/t.co\/xy6Kvhob0Z"

SENDING TO S3 BUCKET: "id":674791877900636160|"text":"#David_Bell #Silicon_Valley
- #NASA's not so keen to talk about quantum ... - https:\/\/t.co\/nedFEKKoIr
https:\/\/t.co\/wACh45PM63"

SENDING TO S3 BUCKET: "id":674792018871062528|"text":"1 hora en #Beijing es como
estar un d\u00eda en la Av. Abancay. cof cof!\n#beijingpollution  https:\/\/t.co\/
sRxk0cByQP"

SENDING TO S3 BUCKET: "id":674792040283152384|"text":"RT @megsalamode: Stunning
footage of #AtlasV launch with #Cygnus #OA4 in 4K Ultra-High Def \ud83d\ude0d
https:\/\/t.co\/XyGMnOS2rf #UHD #NASA https:\/\/t.c\u2026"
```

(etc.)

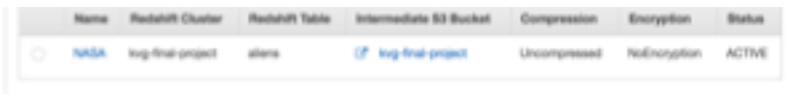
The Kinesis management console shows that our Firehose delivery stream has been created:



Delivery Streams

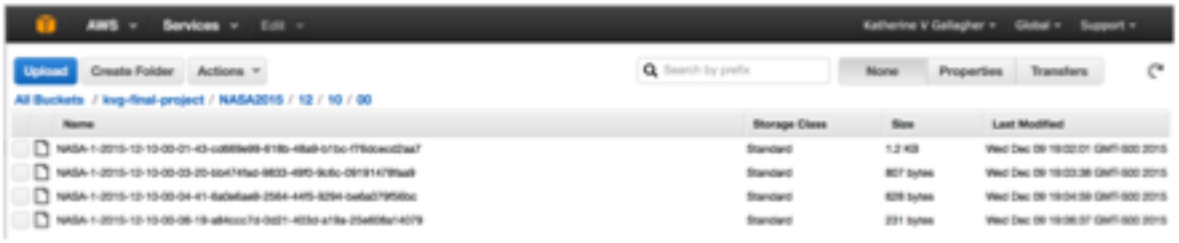
Redshift Delivery Streams

	Name	Redshift Cluster	Redshift Table	Intermediate S3 Bucket	Compression	Encryption	Status



	Name	Redshift Cluster	Redshift Table	Intermediate S3 Bucket	Compression	Encryption	Status
○	NASA	kvg-final-project	aliens	kvg-final-project	Uncompressed	NoEncryption	ACTIVE

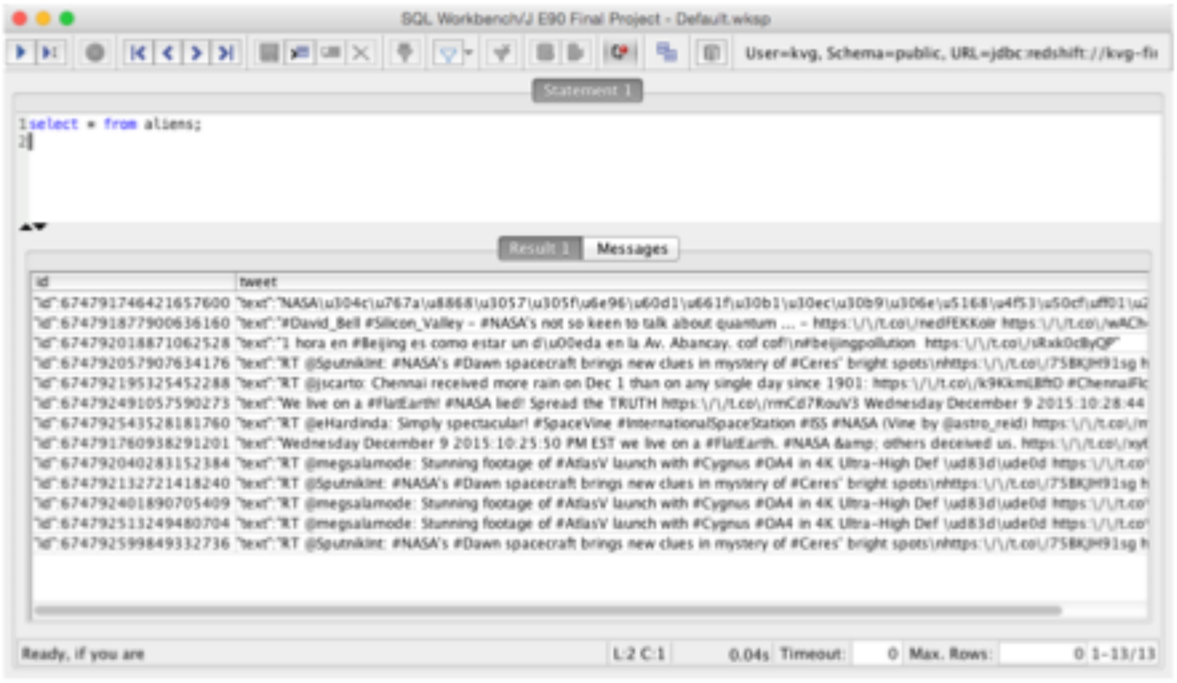
We can check our S3 bucket and see the tweets arriving:



All Buckets / kvg-final-project / NASA2015 / 12 / 10 / 00

Name	Storage Class	Size	Last Modified
NASA-1-2015-12-10-00-01-43-cd669e99-618b-48a9-b1bc-f76dcecd2aa7	Standard	1.2 KB	Wed Dec 09 19:02:01 GMT-500 2015
NASA-1-2015-12-10-00-03-20-bb474fad-9833-49f0-9c8c-09191478faa9	Standard	807 bytes	Wed Dec 09 19:03:38 GMT-500 2015
NASA-1-2015-12-10-00-04-41-8a0e6ae9-2584-44f5-8294-be6a079f56bc	Standard	828 bytes	Wed Dec 09 19:04:59 GMT-500 2015
NASA-1-2015-12-10-00-06-19-a84ccc7d-0d21-403d-a18a-25e608a14079	Standard	231 bytes	Wed Dec 09 19:06:37 GMT-500 2015

And SQL shows the tweets being properly indexed in our Redshift "aliens" table:

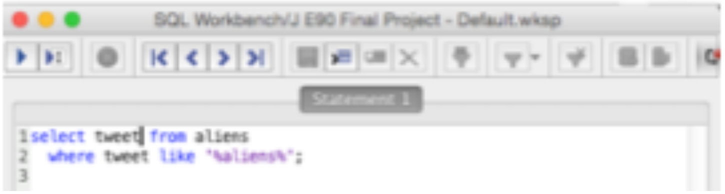

SQL Workbench/J E90 Final Project - Default.wksp
User=kvg, Schema=public, URL=jdbc:redshift://kvg-fi
Statement 1
Result 1
Messages

```
select * from aliens;
```

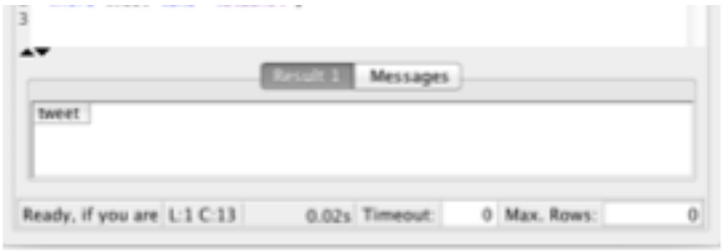
```
id                      tweet
"id":674791746421657600 "text":"NASA\u304c\u767a\u8868\u3057\u305f\u6e96\u60d1\u661f\u30b1\u30ec\u30b9\u306e\u5168\u4f53\u50cf\uff01\u2
"id":674791877900636160 "text":"#David_Bell #Silicon_Valley - #NASA's not so keen to talk about quantum ... - https:\/\/t.co\/nedfEKKoIr https:\/\/t.co\/wACh
"id":674792018871062528 "text":"1 hora en #Beijing es como estar un d\u00eda en la Av. Abancay. cof cof!\n#beijingpollution  https:\/\/t.co\/sRxk0cByQF"
"id":674792057907634176 "text":"RT @SputnikInt: #NASA's #Dawn spacecraft brings new clues in mystery of #Ceres' bright spots\nhttps:\/\/t.co\/7SBKJH91sg h
"id":674792195325452288 "text":"RT @jscarto: Chennai received more rain on Dec 1 than on any single day since 1901: https:\/\/t.co\/k9KkmLBftD #ChennaiFlc
"id":674792491057590273 "text":"We live on a #FlatEarth! #NASA lied! Spread the TRUTH https:\/\/t.co\/rmCd7RouV3 Wednesday December 9 2015:10:28:44
"id":674792543528181760 "text":"RT @eHardinda: Simply spectacular! #SpaceVine #InternationalSpaceStation #ISS #NASA (Vine by @astro_reid) https:\/\/t.co\/m
"id":674791760938291201 "text":"Wednesday December 9 2015:10:25:50 PM EST we live on a #FlatEarth. #NASA &amp; others deceived us. https:\/\/t.co\/xyt
"id":674792040283152384 "text":"RT @megsalamode: Stunning footage of #AtlasV launch with #Cygnus #OA4 in 4K Ultra-High Def \ud83d\ude0d https:\/\/t.co\
"id":674792132721418240 "text":"RT @SputnikInt: #NASA's #Dawn spacecraft brings new clues in mystery of #Ceres' bright spots\nhttps:\/\/t.co\/7SBKJH91sg h
"id":674792401890705409 "text":"RT @megsalamode: Stunning footage of #AtlasV launch with #Cygnus #OA4 in 4K Ultra-High Def \ud83d\ude0d https:\/\/t.co\
"id":674792513249480704 "text":"RT @megsalamode: Stunning footage of #AtlasV launch with #Cygnus #OA4 in 4K Ultra-High Def \ud83d\ude0d https:\/\/t.co\
"id":674792599849332736 "text":"RT @SputnikInt: #NASA's #Dawn spacecraft brings new clues in mystery of #Ceres' bright spots\nhttps:\/\/t.co\/7SBKJH91sg h
```

```
Ready, if you are     L:2 C:1     0.04s  Timeout:  0  Max. Rows:  0  1-13/13
```

Although one of the perks of Firehose is that you don't actually *have* to analyze your data - Firehose is merely a simple, no fuss delivery stream - I executed a search in Redshift to see how many of 100 #NASA tweets contained the term "aliens". The result was sadly identical to that of the global search for extra-terrestrials: 0 aliens.

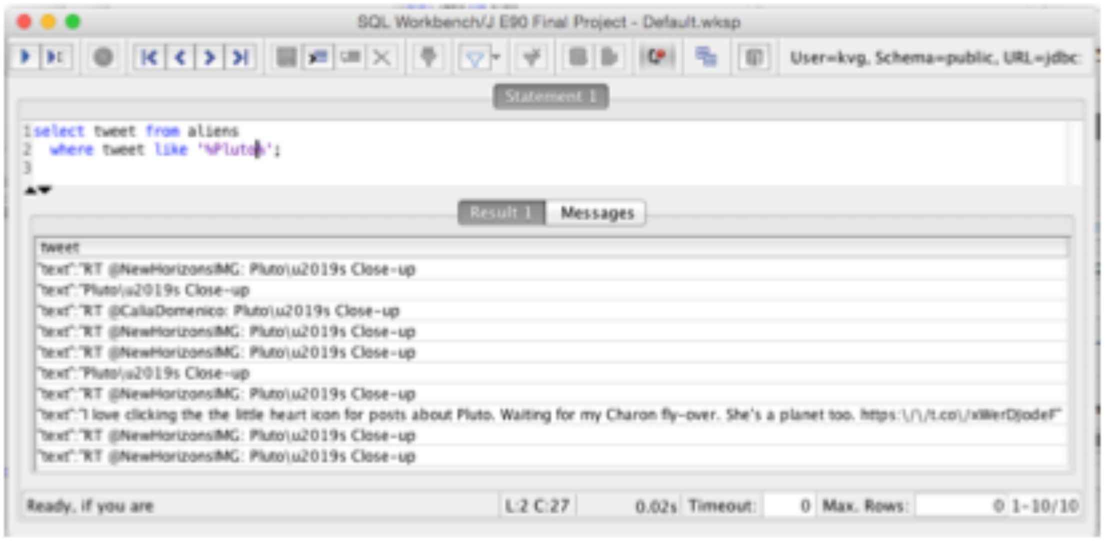


```
select tweet from aliens
  where tweet like '%aliens%';
```



```
tweet

Ready, if you are  L:1 C:13   0.02s  Timeout:  0  Max. Rows:  0
```

However, I was pleased to find 10 tweets paying homage to NASA's *New Horizons* as it journeys past Pluto.



SQL Workbench/J E90 Final Project - Default.wksp
User=kvg, Schema=public, URL=jdbc:
Statement 1
Result 1
Messages

```
select tweet from aliens
  where tweet like '%Pluto%';
```

```
tweet
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up
"text":"Pluto\u2019s Close-up
"text":"RT @CallaDomenico: Pluto\u2019s Close-up
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up
"text":"Pluto\u2019s Close-up
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up
"text":"I love clicking the the little heart icon for posts about Pluto. Waiting for my Charon fly-over. She's a planet too. https:\/\/t.co\/xWerDjodeF"
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up
"text":"RT @NewHorizonsIMG: Pluto\u2019s Close-up

Ready, if you are   L:2 C:27   0.02s  Timeout:  0  Max. Rows:  0 1-10/10
```