IBM WebSphere
Technical University 2014

28 - 31 October 2014 | Düsseldorf, Germany

Developing a first application with IBM MQ Light:

Lab Instructions

Lab Username = demo

Lab Password = “sample”

Authors:

Steve Upton, MQ Light Development Team

Matthew Whitehead, MQ Light Development Team

Rob Nicholson, STSM Application Messaging

What you will learn

This lab will teach you how you can improve the responsiveness and scalability of your web applications, both on premise and in the cloud.

It will explain how to off-load heavy workloads to separate worker threads while your web handlers deal quickly and efficiently with the requests from your online users.

As software developers we want our applications to be responsive and scalable to really engage users, but it's not always easy to write code that behaves like this. MQ Light and the MQ Light Service in BlueMix are great tools that helps applications off-load work to be dealt with asynchronously thus ensuring your applications responds quickly. Additionally, as workload increases, applications that use MQ Light become very easy to scale up.

What the lab covers

This lab has 5 parts to it:

1. Running the sample node.js application
2. Improving the sample application by separating the web-facing component from the data-processing component
3. Scaling up the number of data-processing threads to cope with the workload
4. Deploying your finished application to IBM BlueMix to run it in the cloud using the MQ Light service
5. Changing the worker application from node.js to Java using JMS – illustrating polyglot messaging.

Setup – Extract the source for the labs

1. Open a terminal window (under *the Applications->System Tools* menu at the top of the screen) and navigate to */home/demo/mql/source*

2. Extract the source files from git by typing “*./extract-source.sh*” and hitting enter.

This should extract the 4 projects that we will be using to run the lab into the source directory.

> **Note:** If step 2 failed, for example because of network problems, there is a backup copy of the source folders in */home/demo/mql/misc/local-git-repos*
>
> Navigate to */home/demo/mql/misc/local-git-repos* in the file browser

3. If you wish to run the BlueMix part of the lab (parts 4 and 5) you will need to get a BlueMix ID. We would suggest you skip straight to the first steps in part 4 where there are instructions on signing up for BlueMix. That will give us more time to make sure you have a working BlueMix ID by the time you come to run those parts of the lab.

When you have requested your BlueMix ID you can return to part 1 of the lab.

You should now be ready to run the lab!

Part 1 – Running the sample node.js application

To introduce MQ Light and demonstrate how it can be incorporated into your projects we have constructed a node.js application which will process a stream of data from Twitter. In our sample scenario, the messages arriving from Twitter are used in place of messages being submitted by web users.

In the sample application, when messages are received from Twitter they are processed by a function running in the same thread that received the message. The processing performs some basic analysis of the data to simulate the sort of backend processing your applications might do.

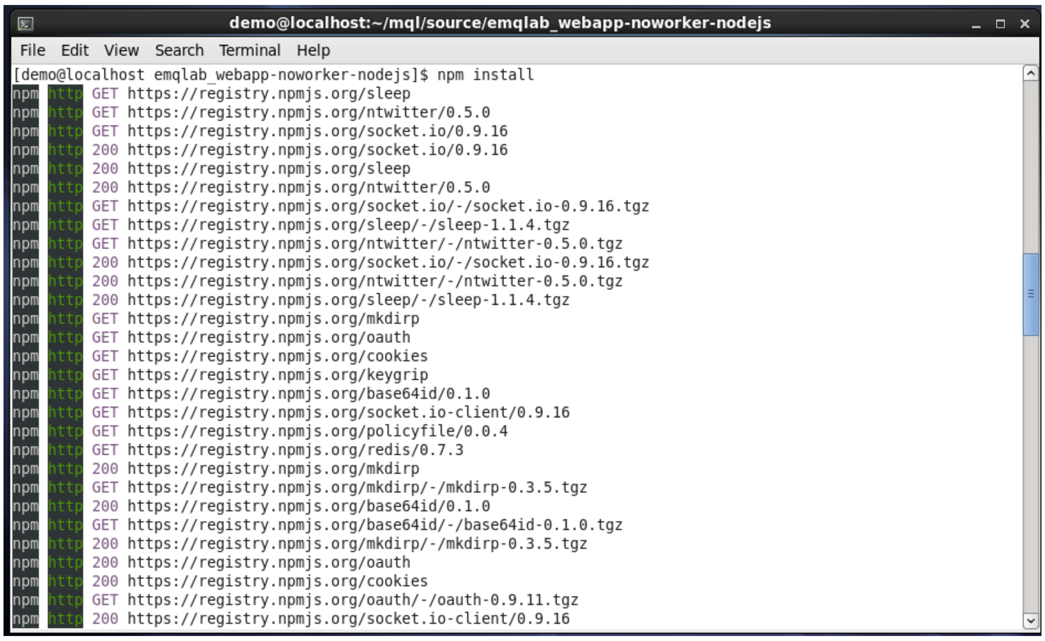
In this first part of the lab you will start the sample application and see messages arriving from Twitter and being processed.

1. Copy the twitter API keys and credentials into the applications.

Access to the twitter APIs that we will be using in this lab requires API keys. If you are running this lab during the scheduled lab session at WTU then the lab source that you downloaded from github will contain an active set of API keys and you can move on to step 2. If you are running this lab at any other time then you will need to follow the procedure in Appendix A to obtain twitter API keys.

2. Install the node.js dependencies

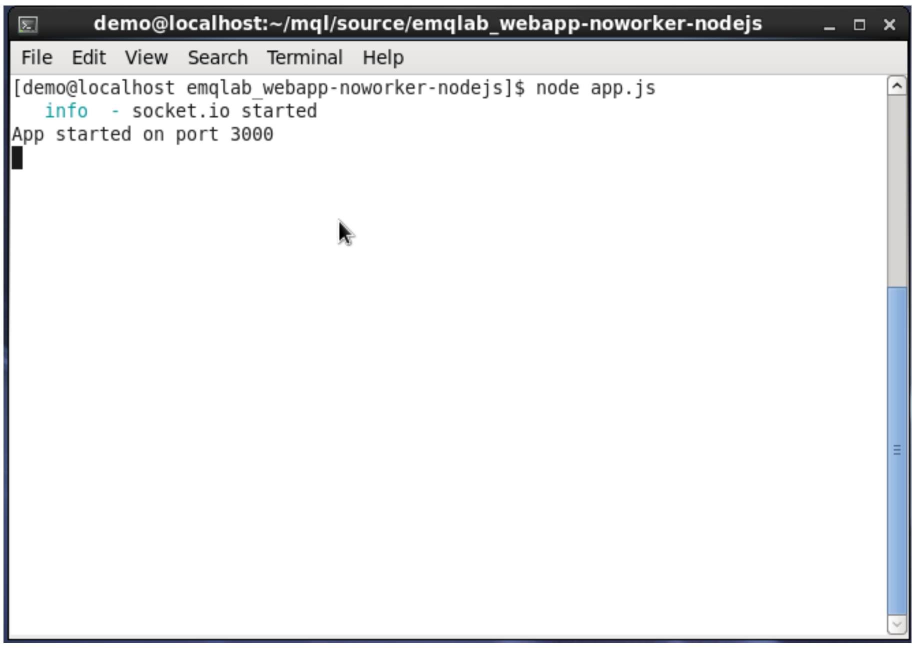
Before you can run the sample we need to install the dependencies the application has. Open a terminal window and navigate to */home/demo/mql/source/webapp-noworker.nodejs.* Type the command "*npm install*" and hit enter:


demo@localhost:~/mql/source/emqlab_webapp-noworker-nodejs
File Edit View Search Terminal Help

```
[demo@localhost emqlab_webapp-noworker-nodejs]$ npm install
npm http GET https://registry.npmjs.org/sleep
npm http GET https://registry.npmjs.org/ntwitter/0.5.0
npm http GET https://registry.npmjs.org/socket.io/0.9.16
npm http 200 https://registry.npmjs.org/socket.io/0.9.16
npm http 200 https://registry.npmjs.org/sleep
npm http 200 https://registry.npmjs.org/ntwitter/0.5.0
npm http GET https://registry.npmjs.org/socket.io/-/socket.io-0.9.16.tgz
npm http GET https://registry.npmjs.org/sleep/-/sleep-1.1.4.tgz
npm http GET https://registry.npmjs.org/ntwitter/-/ntwitter-0.5.0.tgz
npm http 200 https://registry.npmjs.org/socket.io/-/socket.io-0.9.16.tgz
npm http 200 https://registry.npmjs.org/ntwitter/-/ntwitter-0.5.0.tgz
npm http 200 https://registry.npmjs.org/sleep/-/sleep-1.1.4.tgz
npm http GET https://registry.npmjs.org/mkdirp
npm http GET https://registry.npmjs.org/oauth
npm http GET https://registry.npmjs.org/cookies
npm http GET https://registry.npmjs.org/keygrip
npm http GET https://registry.npmjs.org/base64id/0.1.0
npm http GET https://registry.npmjs.org/socket.io-client/0.9.16
npm http GET https://registry.npmjs.org/policyfile/0.0.4
npm http GET https://registry.npmjs.org/redis/0.7.3
npm http 200 https://registry.npmjs.org/mkdirp
npm http GET https://registry.npmjs.org/mkdirp/-/mkdirp-0.3.5.tgz
npm http 200 https://registry.npmjs.org/base64id/0.1.0
npm http GET https://registry.npmjs.org/base64id/-/base64id-0.1.0.tgz
npm http 200 https://registry.npmjs.org/mkdirp/-/mkdirp-0.3.5.tgz
npm http 200 https://registry.npmjs.org/oauth
npm http 200 https://registry.npmjs.org/cookies
npm http GET https://registry.npmjs.org/oauth/-/oauth-0.9.11.tgz
npm http 200 https://registry.npmjs.org/socket.io-client/0.9.16
```

3. Start the node.js application.

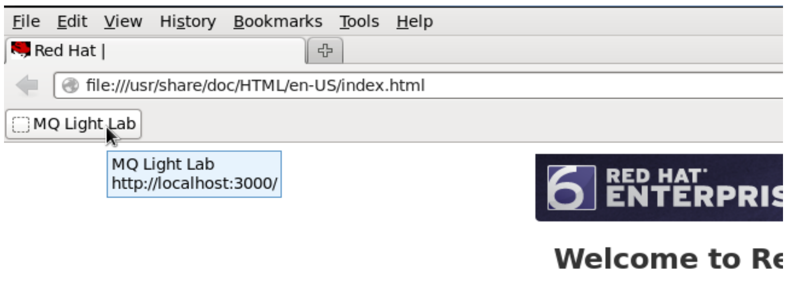
Type *node app.js* and hit enter:



demo@localhost:~/mql/source/emqlab_webapp-noworker-nodejs
File Edit View Search Terminal Help

```
[demo@localhost emqlab_webapp-noworker-nodejs]$ node app.js
   info  - socket.io started
App started on port 3000
```

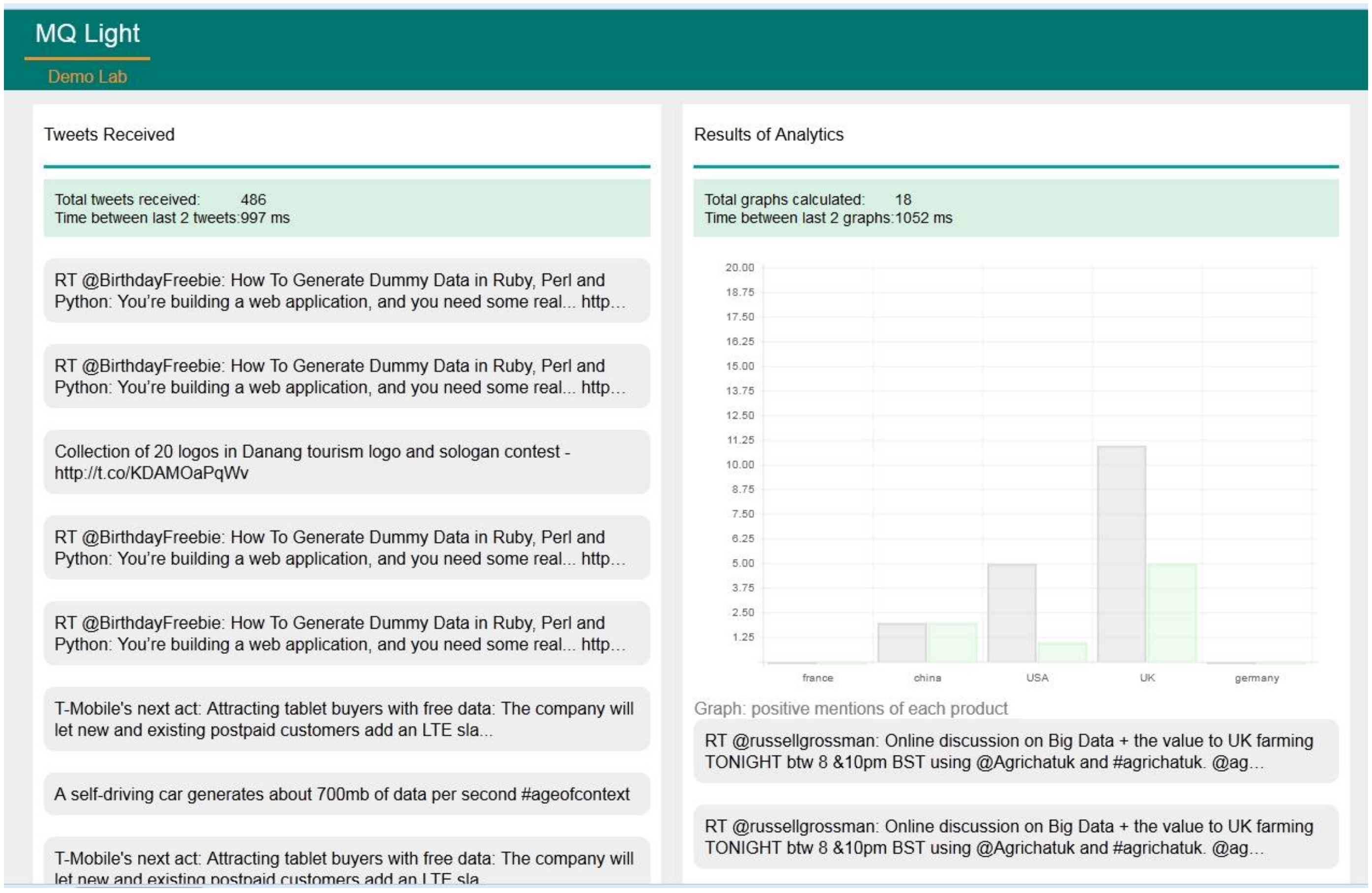
You will notice that nothing much happens. That's because the sample application waits for a browser to connect before it starts collecting data from Twitter. To start the application running, start the firefox web browser, open a new tab in the browser and open the MQ Light Lab bookmark on the bookmarks toolbar. This tab opens the URL: http://localhost:3000/



File Edit View History Bookmarks Tools Help
Red Hat |
file:///usr/share/doc/HTML/en-US/index.html
MQ Light Lab
MQ Light Lab
http://localhost:3000/
RED HAT ENTERPRIS
Welcome to Re

:

The web page should look something like this:


MQ Light

Demo Lab

Tweets Received

Total tweets received: 486
Time between last 2 tweets:997 ms

RT @BirthdayFreebie: How To Generate Dummy Data in Ruby, Perl and Python: You're building a web application, and you need some real... http...

RT @BirthdayFreebie: How To Generate Dummy Data in Ruby, Perl and Python: You're building a web application, and you need some real... http...

Collection of 20 logos in Danang tourism logo and sologan contest - http://t.co/KDAMOaPqWv

RT @BirthdayFreebie: How To Generate Dummy Data in Ruby, Perl and Python: You're building a web application, and you need some real... http...

RT @BirthdayFreebie: How To Generate Dummy Data in Ruby, Perl and Python: You're building a web application, and you need some real... http...

T-Mobile's next act: Attracting tablet buyers with free data: The company will let new and existing postpaid customers add an LTE sla...

A self-driving car generates about 700mb of data per second #ageofcontext

T-Mobile's next act: Attracting tablet buyers with free data: The company will let new and existing postpaid customers add an LTE sla

Results of Analytics

Total graphs calculated: 18
Time between last 2 graphs:1052 ms

Graph: positive mentions of each product

RT @russellgrossman: Online discussion on Big Data + the value to UK farming TONIGHT btw 8 &10pm BST using @Agrichatuk and #agrichatuk. @ag...

RT @russellgrossman: Online discussion on Big Data + the value to UK farming TONIGHT btw 8 &10pm BST using @Agrichatuk and #agrichatuk. @ag...

Tweets should start appearing on the left hand side of the screen. On the right hand side of the screen is a graph of tweets that mention the "products" we are interested in. For the purposes of this lab we have replaced the product names with the names of countries to ensure a good flow of matching tweets. Whenever a new tweet arrives from the twitter API, the application scans the tweet to determine if it matches a "product". If it does then the tweet will appear on the right hand side and the gray bar for that product is updated with the cumulative total of matching tweets. If the tweet is analysed to reflect a positive sentiment then the height of green bar of positive tweets is also increased.

You will notice that tweets are being processed and displayed at a rate not exceeding one per second. You will also notice that new tweets aren't shown on the left hand side until the graph has finished generating.

The reason for this is because the sentiment analysis and graph generating code is running in the same thread as the code that communicates with Twitter. A new tweet cannot be received until the graph is generated and the application can ask for the next tweet.

If the tweets were actually requests for orders from your customers, every thread handling a customer's request would be locked up waiting for the order processing to complete. The thread would not be available to service another customer's request.

4. Stop the application and clear your browser

Terminate the app.js application in the terminal by pressing Ctrl-C in the terminal window.

Hit refresh on the browser window to clear it and prevent the web page from automatically connecting to the applications we run later in the lab.

Part 2 – Improving the sample application by separating the web-facing component from the data-processing component

In part 1 we ran a single-threaded application which handled receiving tweets and processing them all in the same thread.

This meant that the number of tweets we could receive and handle was limited to the rate that we could process them.

In this part of the lab we are going to separate the code that receives the tweets from the code that processes them. This will allow us in the later parts of the lab to scale the number of threads we have processing the tweets to allow the application to cope with more workload. We will be using MQ Light as the elastic buffer between the web application and the worker processes.

There are 2 applications you will need for this section.

- webapp-offload.nodejs/app.js
    - This does a similar thing to the original app.js which we used earlier. We've removed the logic that processed the tweets and moved it into worker.js (see below). We've also added the code required to connect it to an MQ Light server. This will be necessary for offloading the tweets to MQ Light.

- worker.nodejs/worker.js
    - The worker application is a new application that we didn't need in part 1 because all of the processing was done in one place. The worker has the same code that the original app.js used to process tweets, but it also has the code required to connect it to MQ Light and consumer the messages that have been offloaded by webapp-offload.nodejs/app.js

1. Start MQ Light

Because we're going to start offloading work to MQ Light, we need to start the MQ Light runtime so that the applications can connect to it.

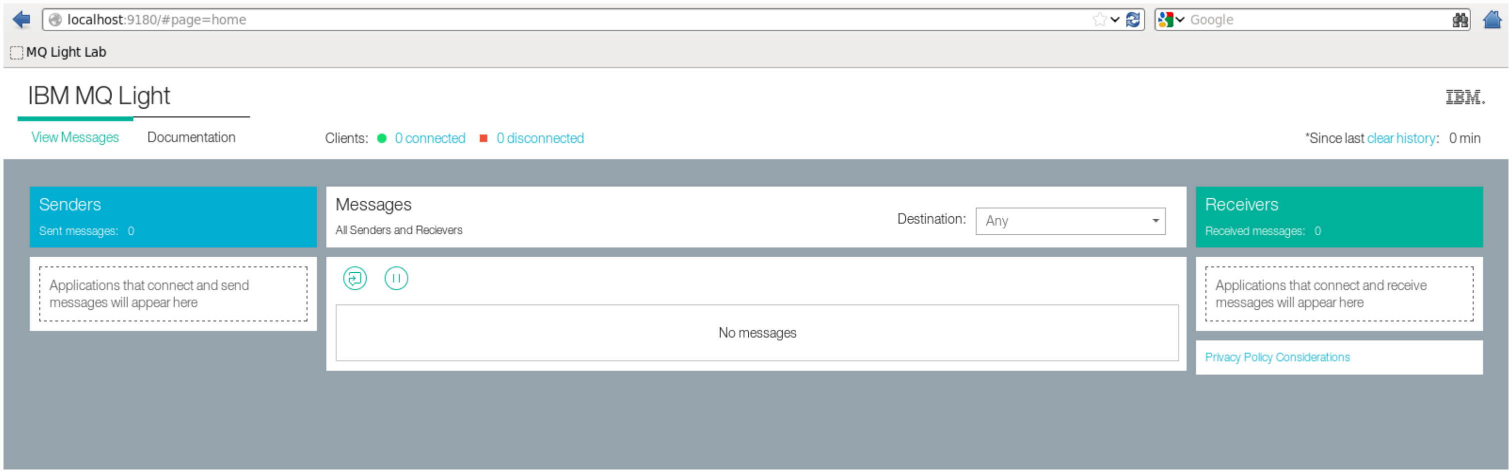
In the mql folder (/home/demo/mql) there is a folder called mqlight-developer-1.0 . This contains MQ Light Version 1.0.

Note: If you want to try things out on your own machine after this lab session you can download MQ Light from our website at https://www.ibmdw.net/messaging/mq-light/

To start MQ Light, double click on “Start IBM MQ Light”. A new terminal window will open and you will be asked to accept the license. Press 1 and hit enter. Next you will be asked if you wish to enable passwords security and ssl. In each case Press N and hit enter.
MQ Light will start up.

2. Checkout the MQ Light GUI

When MQ Light has finished starting it will automatically start the GUI in the browser. The first page you will see is the documentation tab. To switch to the development screen click on “View Messages” at the top left of the page. You should now see something like this:


localhost:9180/#page=home

MQ Light Lab

IBM MQ Light

View Messages | Documentation

Clients: ● 0 connected ■ 0 disconnected

*Since last clear history: 0 min

Senders
Sent messages: 0

Applications that connect and send messages will appear here

Messages
All Senders and Recievers

Destination: Any

No messages

Receivers
Received messages: 0

Applications that connect and receive messages will appear here

Privacy Policy Considerations

We will explore the UI in more detail as we go through running the different parts of the application.

3. Open the application to inspect the calls it makes to connect to MQ Light:

If you open *home/demo/mql/source/webapp-offload.nodejs/app.js* in the text editor you can see the code we've added to allow the application to connect to MQ Light and offload the tweets for processing. You may want to refer to the MQ Light API documentation at https://www.npmjs.org/package/mqlight as we examine the code.

```
var mqlight = require('mqlight');
```

This line at the top of the file simply loads the mqlight node.js libraries.

```
var id='WEB_' + uuid.v4().substring(0, 7);
```

This line creates a unique client ID which starts “WEB_” . The client libraries can create a unique ID for us but we want to be able to recognise connections from the web front end.

```
if (process.env.VCAP_SERVICES) {
  // App is running in Bluemix.
  // >>> BLUEMIX CODE REMOVED FOR CLARITY <<<
} else
    // App is running outside of Bluemix

    opts = {  service:'amqp://localhost:5672',id:id};
}
```

These lines set up the options for connecting to the mqlight service. When running locally mq light listens on port 5672 on localhost using the AMQP prorotcol.

```
  var client = mqlight.createClient(opts);
```

This line creates an instance of the client. Note that the client does not need an explicit connect or start method invocation. It starts automatically.

```
client.on('started', function() { . . . }
```

Here we are telling the client which function should be called when a connection to MQ Light has been successfully started.

```
function sendMessage(topic, body) {
            client.send(topic, body, function(err, msg) { . . .});
}
```

When a tweet has been received, this function is called to send line tells the client to send a message to MQ Light on the specified topic. Again we pass in a callback function which will be called when the message has been sent, or if an error occurred.

```
var destination = client.subscribe('processedData', callback);
```

The final line we'll look at subscribes the client to a destination with a topicpattern ‘processedData’ . This pattern matches the topic to which the worker threads will send their responses. When a message is received by this destination, the callback method will be invoked.

The worker.js application uses similar calls to subscribe to tweet messages sent to it by the offloading application and send responses back to it.

Let's try running them!

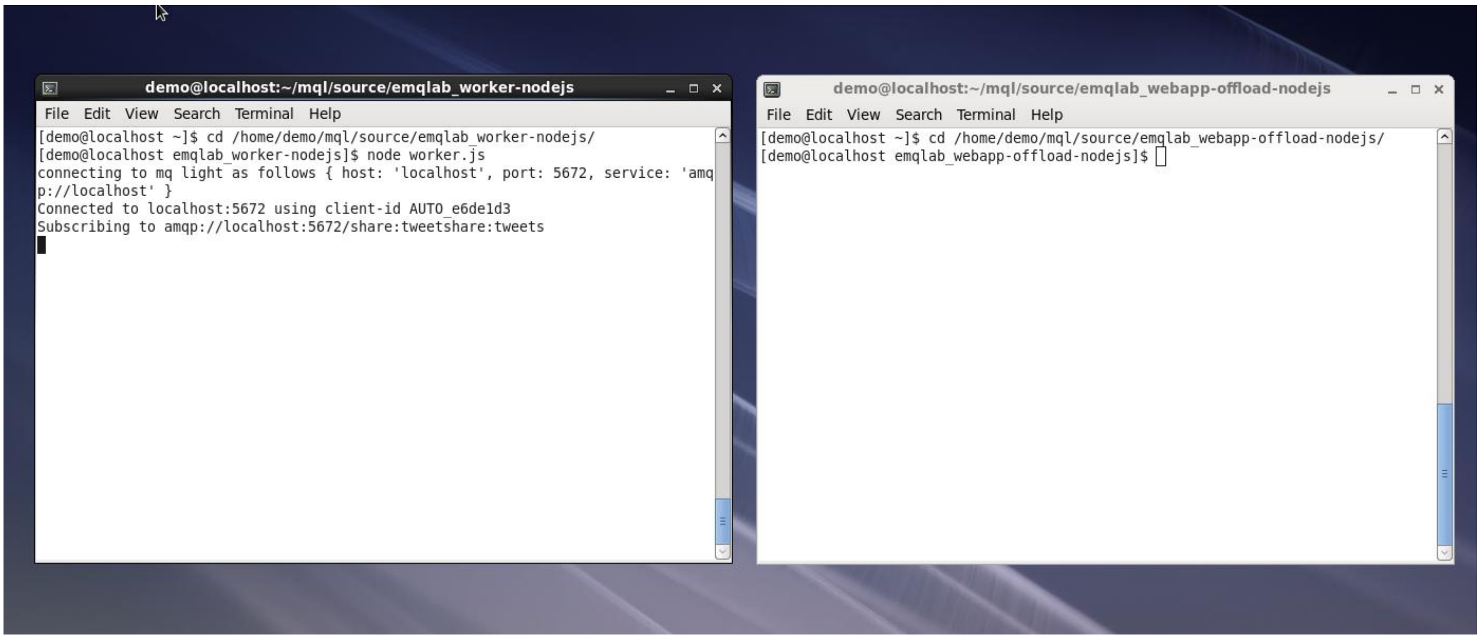
4. Make sure you have 2 terminal windows open. In one of them navigate to */home/demo/mql/source/worker.nodejs* and in the other one navigate to */home/demo/mql/source/webapp-offload.nodejs*

As before, we need to install the node.js dependencies for each application.

Type *npm install* into each terminal and press enter.

5. In the first terminal run the command "*node worker.js*"

The worker application will connect to MQ Light and wait for messages to arrive:

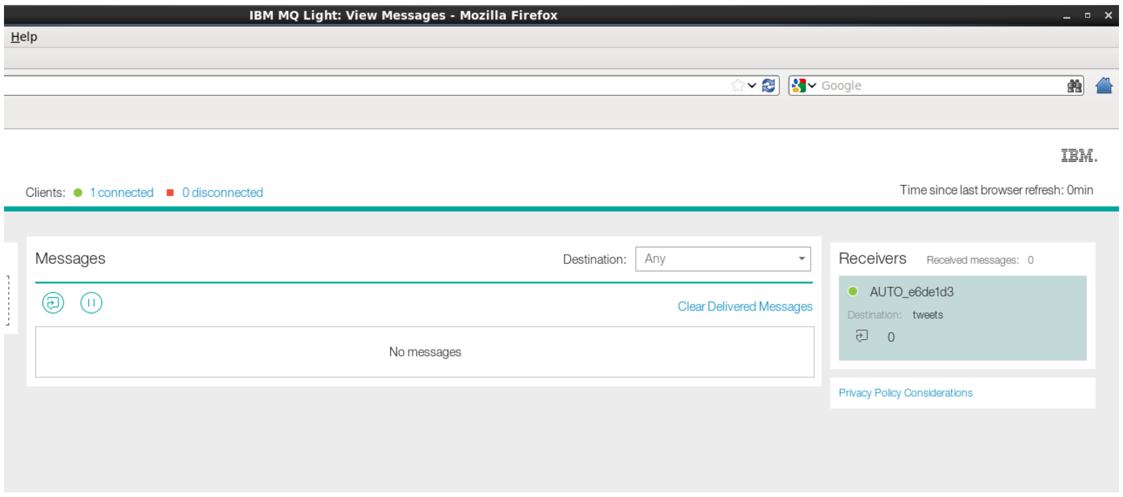

demo@localhost:~/mql/source/emqlab_worker-nodejs
File Edit View Search Terminal Help
demo@localhost:~/mql/source/emqlab_webapp-offload-nodejs
File Edit View Search Terminal Help

```
[demo@localhost ~]$ cd /home/demo/mql/source/emqlab_worker-nodejs/
[demo@localhost emqlab_worker-nodejs]$ node worker.js
connecting to mq light as follows { host: 'localhost', port: 5672, service: 'amq
p://localhost' }
Connected to localhost:5672 using client-id AUTO_e6de1d3
Subscribing to amqp://localhost:5672/share:tweetshare:tweets
```

```
[demo@localhost ~]$ cd /home/demo/mql/source/emqlab_webapp-offload-nodejs/
[demo@localhost emqlab_webapp-offload-nodejs]$
```

If you switch back to the GUI in your browser you should now see that the worker application has connected successfully, and that it is in the receivers view on the right hand side of the screen because it has subscribed to a destination to receive messages from.

You can see that the the worker application has subscribed to a destination called 'tweets'.


IBM MQ Light: View Messages - Mozilla Firefox
Help
Google
IBM.
Clients: 1 connected 0 disconnected
Time since last browser refresh: 0min
Messages
Destination: Any
Clear Delivered Messages
No messages
Receivers Received messages: 0
AUTO_e6de1d3
Destination: tweets
0
Privacy Policy Considerations

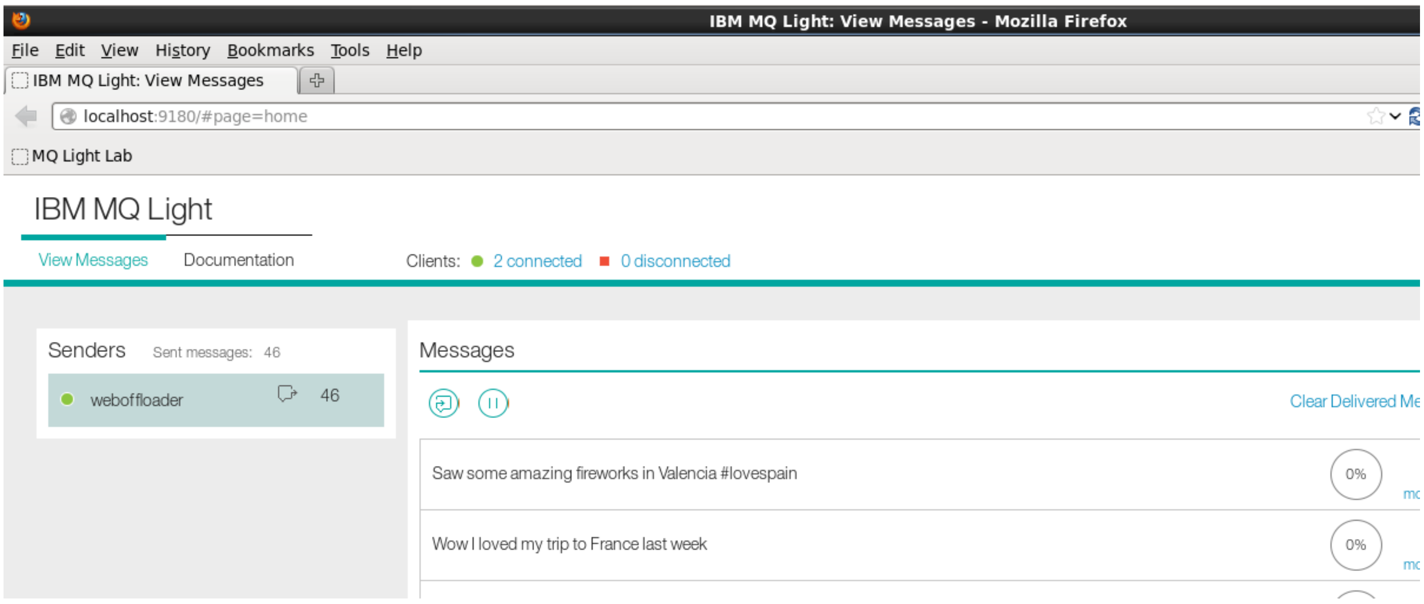
The MQ Light GUI is designed to make development of apps very quick and easy. When you're using MQ Light to develop your own applications you can use the GUI to check that your app is connecting to MQ Light correctly, and to examine any destinations that the clients subscribe to.

6. In the second terminal run the command “*node app.js*”

The application will start and, as before, it will wait for the browser to connect to it before it begins to receive data from Twitter.

7. In your browser open a new tab and navigate back to the MQ Light demo page using the bookmark on the toolbar. The browser connects to the app.js application which will now start receiving messages from Twitter.

Now that the application has started properly, you can use the MQ Light GUI to check that it is connected to MQ Light and that it is publishing messages to the worker application. If everything is working correctly you should see the application in the list of senders on the left hand side of the screen:


IBM MQ Light: View Messages - Mozilla Firefox

localhost:9180/#page=home

IBM MQ Light

Clients: ● 2 connected ■ 0 disconnected

Senders Sent messages: 46

weboffloader 46

Messages

Saw some amazing fireworks in Valencia #lovespain 0%

Wow I loved my trip to France last week 0%

File Edit View History Bookmarks Tools Help
IBM MQ Light: View Messages
MQ Light Lab
View Messages
Documentation
Clear Delivered Me

Switch back to the “MQ Light Lab” tab in your browser. Notice how the left hand side of the screen is updating as soon as new tweets arrive. This is because instead of processing each tweet it simply hands it off to MQ Light and goes back to receive and display the next one.

The worker thread is asynchronously working through the list of messages arriving on the 'tweet' topic. When it has finished processing a tweet, if the tweet contained a reference to a product name it sends a response to the browser with the result of the sentiment analysis. Any tweets which don't contain references to products are simply discarded.

We have improved the basic application by separating the steps of receiving tweets and processing them. There is still a problem though. If you at the MQ Light GUI and select the icon to hide completely delivered messages, you will see that the number of unprocessed messages slowly builds up until the point where messages are being discarded because they have exceeded the 15 second time to live that we set.

The reason for this is that although we've freed up the offloading application, we still only have a single worker thread processing the data. The worker thread can only process one tweet per second and if tweets arrive more frequently than that then the worker thread cannot keep up!

In the next part of the lab we will demonstrate how to scale up the number of worker threads which are available to process the tweets.

8. Stop the applications and clear your browser

Terminate the app.js and worker.js applications in their terminals by pressing Ctrl-C in the terminal windows.

Hit refresh on the MQ Light demo browser window to clear it and prevent the web page from automatically connecting to the applications we run later in the lab. You leave the MQ Light GUI running.

Part 3 – Scaling up the number of data-processing threads to cope with the workload

As we saw in the previous section, the single worker thread is unable to cope with the number of tweets being offloaded to it. We can't just make the worker thread go more quickly. What we need to do is create more of them!

The first thing we have to do is change the way the worker application subscribes. In the current version the worker thread subscribes to a private destination with a topic pattern 'tweets'. This means that it receives a copy of every messages published to that topic. If we simply started another instance of the worker application both instances would receive a copy of every tweet. This would mean that each tweet got processed twice which we don't want to do.

To change this behaviour we need both of the worker applications to share the messages between them.   The way we can do this using the MQ Light API is to make the worker threads join a “shared destination”. A shared destination allows a group of applications to share the processing of messages arriving that the destination. Each message arriving at the destination is only given to one of them.

To change the worker application we need to do 2 things:

- Choose a “share name” for the shared destination
- Edit the worker application to make it subscribe to a shared destination.

In these instructions we will use the share group name “tweetprocessors” but you can choose something else if you prefer.

1. Find the worker.js application in the /home/demo/mql/source/worker.nodejs folder, and open it for editing using gedit.

2. Locate the line in the application where it creates the subscription to the 'tweets' topic:

```
client.subscribe('tweets', subOptions, function(err,
                          address) {
```

We need to add another parameter to the subscribe(...) method to specify the name of the share we want our worker to join. Edit the line in the file to look like the following:

```
client.subscribe('tweets', 'tweetprocessors', subOptions, function(err,
                          address) {
```

3. Save the file and close the editor

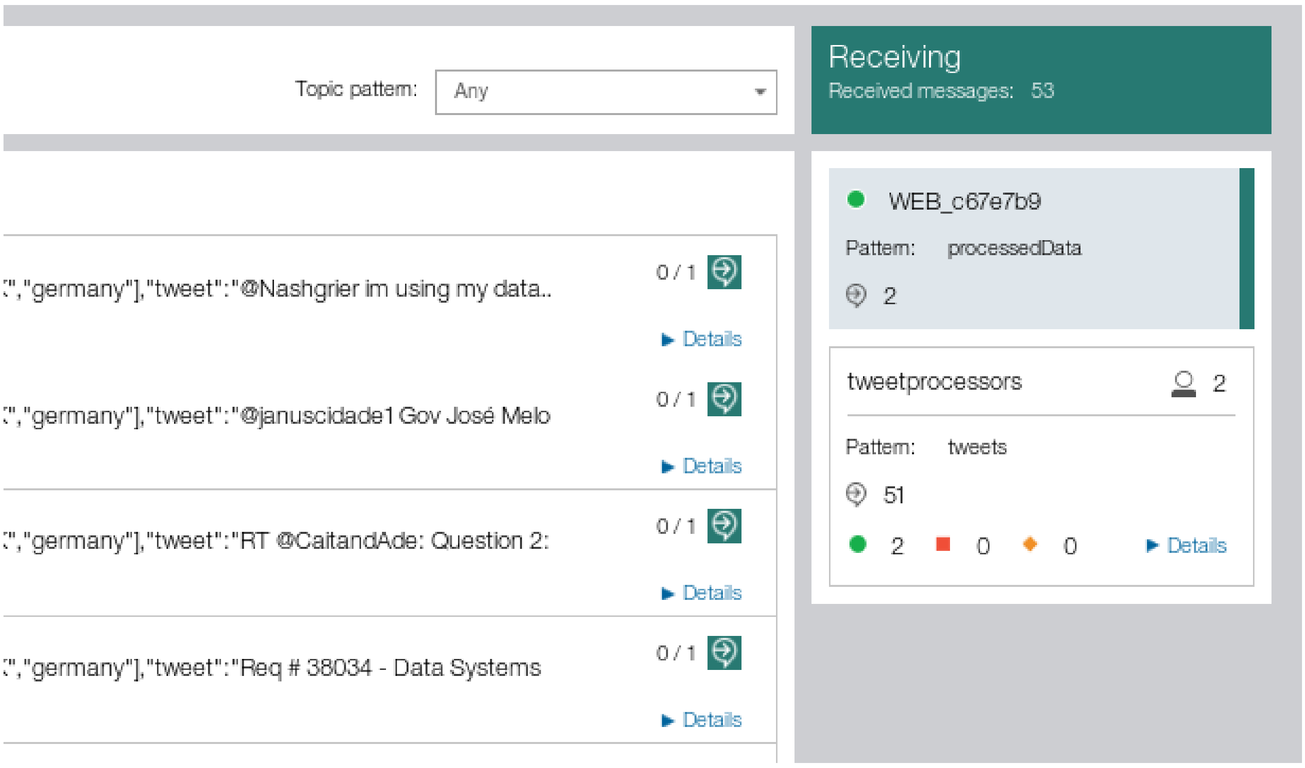
4. Open 2 terminal windows to run the 2 worker applications. In both terminals navigate to */home/demo/mql/source/worker.nodejs*

5. Enter the command *node worker.js* into each terminal and hit enter. This will start both of the worker applications. Open the browser with the MQ Light GUI in it and check that the 2 worker applications are running correctly and connected to MQ Light.

6. In a third terminal window navigate to */home/demo/mql/source/webapp-offload.nodejs* and run "*node* app.js" to start the main application.

As before you must open a browser and launch the MQ Light Lab page using the bookmark on the toolbar. This will start the application consuming messages from Twitter and offloading them to the worker threads via MQ Light.

Now that the application is running, check that the application is connected to MQ Light and publishing messages by opening the MQ Light GUI in the browser. On the right hand side you should see that instead of each individual client having its own private destination there is now a shared destination called “tweetprocessors” .



Topic pattern: Any

:","germany"],"tweet":"@Nashgrier im using my data.. 0 / 1 ▶ Details

:","germany"],"tweet":"@januscidade1 Gov José Melo 0 / 1 ▶ Details

:","germany"],"tweet":"RT @CaitandAde: Question 2: 0 / 1 ▶ Details

:","germany"],"tweet":"Req # 38034 - Data Systems 0 / 1 ▶ Details

Receiving
Received messages: 53

● WEB_c67e7b9
Pattern: processedData
2

tweetprocessors 2
Pattern: tweets
51
● 2 ■ 0 ◆ 0 ▶ Details

You should see that the shared destination has two active receiving connections. To see the client IDs of the receivers click on the “Details” link in the “tweetprocessors” shared destination.

Note: The number of messages received by each worker thread may not be equal. You would expect them to get 1 each in turn, but because of the way MQ Light decides which application is best able to process the work, one instance may process more work than the other.

7. Increase the number of worker threads

Obviously 2 worker threads is better than 1, but we can start as many as we need to cope with the data processing workload. Open a new terminal window and repeat steps 4 and 5 above, checking in the MQ Light GUI that the new worker thread has subscribed to the shared destination and is is running correctly and processing some of the messages.

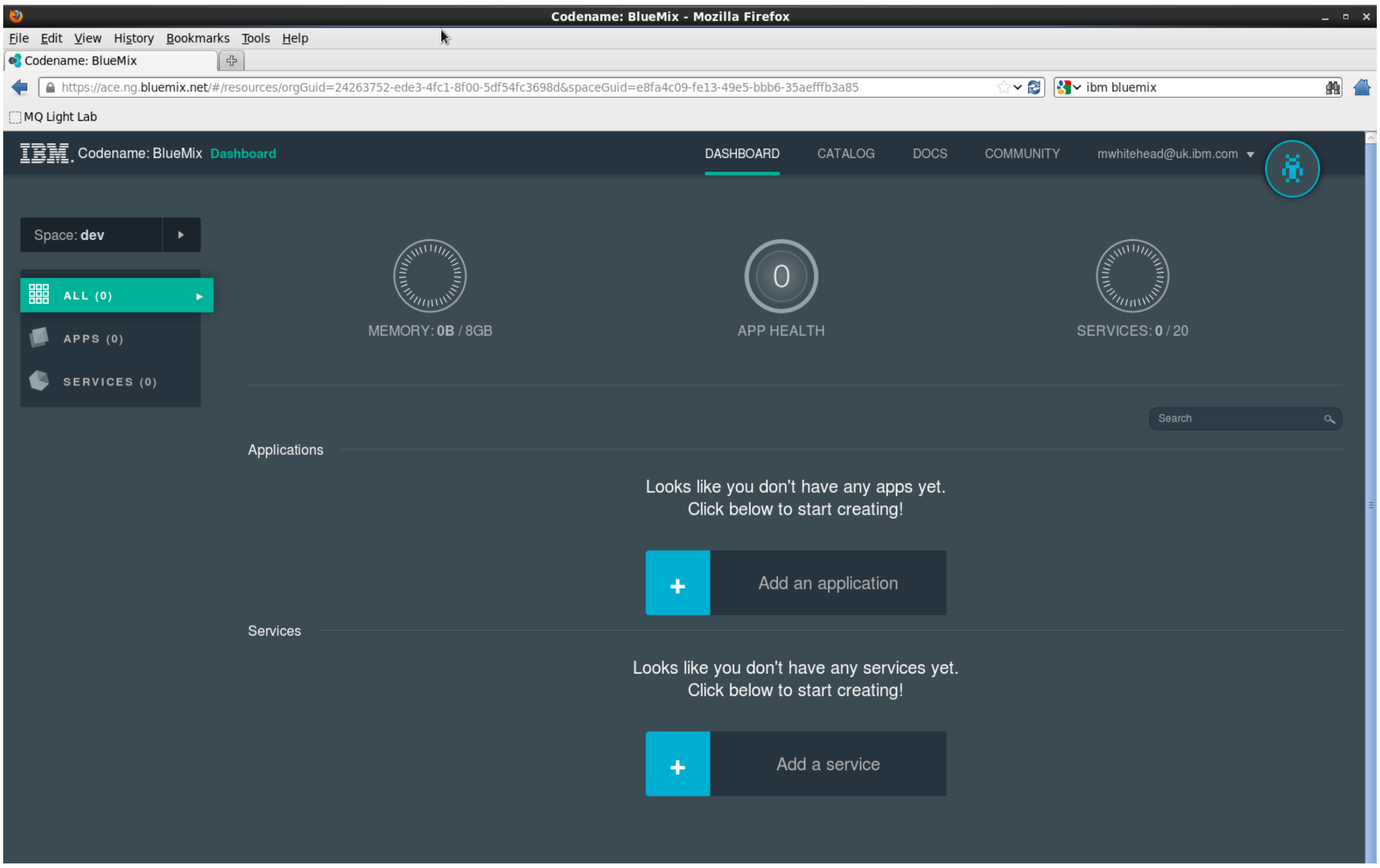
Part 4 – Deploying your finished application to IBM BlueMix to run it in the cloud

You've successfully developed and tested your application using the standalone MQ Light runtime. However, the same API is supported in IBM BlueMix by the MQ Light service.

In this part of the lab we are going to deploy the two applications – offloadapplication.js and worker.js - into IBM BlueMix and bind them both to a single MQ Light service. MQ Light will act in the same way that the local MQ Light runtime did. The applications will connect to it, and offloadapplication.js will publish its messages while the worker.js applications subscribe and receive them.

To do this part of the lab you will need to sign up to IBM BlueMix which you can do here https://ace.ng.bluemix.net/

Once you have signed up and logged in to BlueMix you will be presented with the BlueMix dashboard:


Codename: BlueMix - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Codename: BlueMix
https://ace.ng.bluemix.net/#/resources/orgGuid=24263752-ede3-4fc1-8f00-5df54fc3698d&spaceGuid=e8fa4c09-fe13-49e5-bbb6-35aefffb3a85
ibm bluemix
MQ Light Lab
IBM. Codename: BlueMix Dashboard
DASHBOARD
CATALOG
DOCS
COMMUNITY
mwhitehead@uk.ibm.com
Space: dev
ALL (0)
APPS (0)
SERVICES (0)
MEMORY: 0B / 8GB
APP HEALTH
SERVICES: 0 / 20
Search
Applications
Add an application
Services
Add a service

Looks like you don't have any apps yet.
Click below to start creating!

Looks like you don't have any services yet.
Click below to start creating!

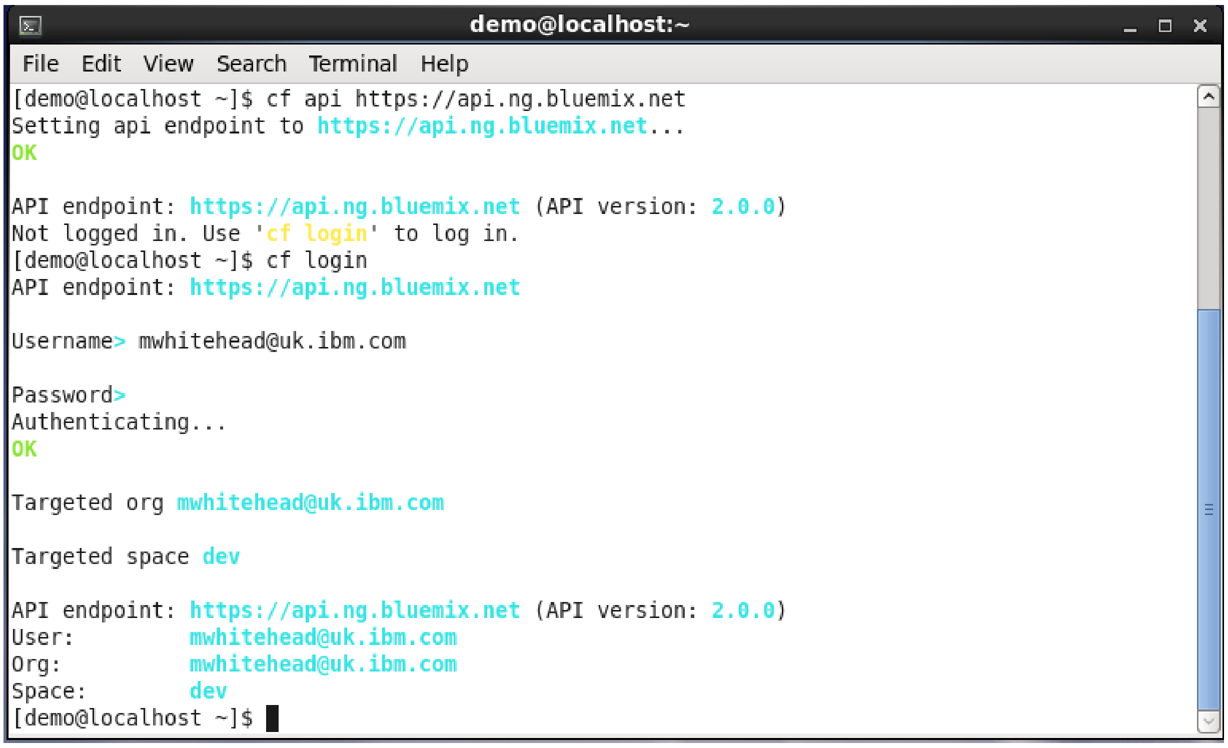
You can define applications and create instances of services using the BlueMix dashboard, but for the rest of the lab we'll use the command line to speed things up a bit. All you need is the username and password you used to sign up to BlueMix. First of all we can try deploying the simple application that doesn't offload any work to MQ Light. This will allow you to familiarise yourself with the commands for deploying applications to BlueMix and check that your credentials are working.

1. Set up your BlueMix command environment to by opening a terminal window and entering the command “*cf api https://api.ng.bluemix.net*”

This tells the cf tool to connect to the BlueMix cloud environment rather than another CloudFoundry environment.

Now enter the command “*cf login*” and press enter.

You will be asked to enter your BlueMix username and password. Once you have logged in you can enter the rest of the cf commands without needing to supply your credentials again.



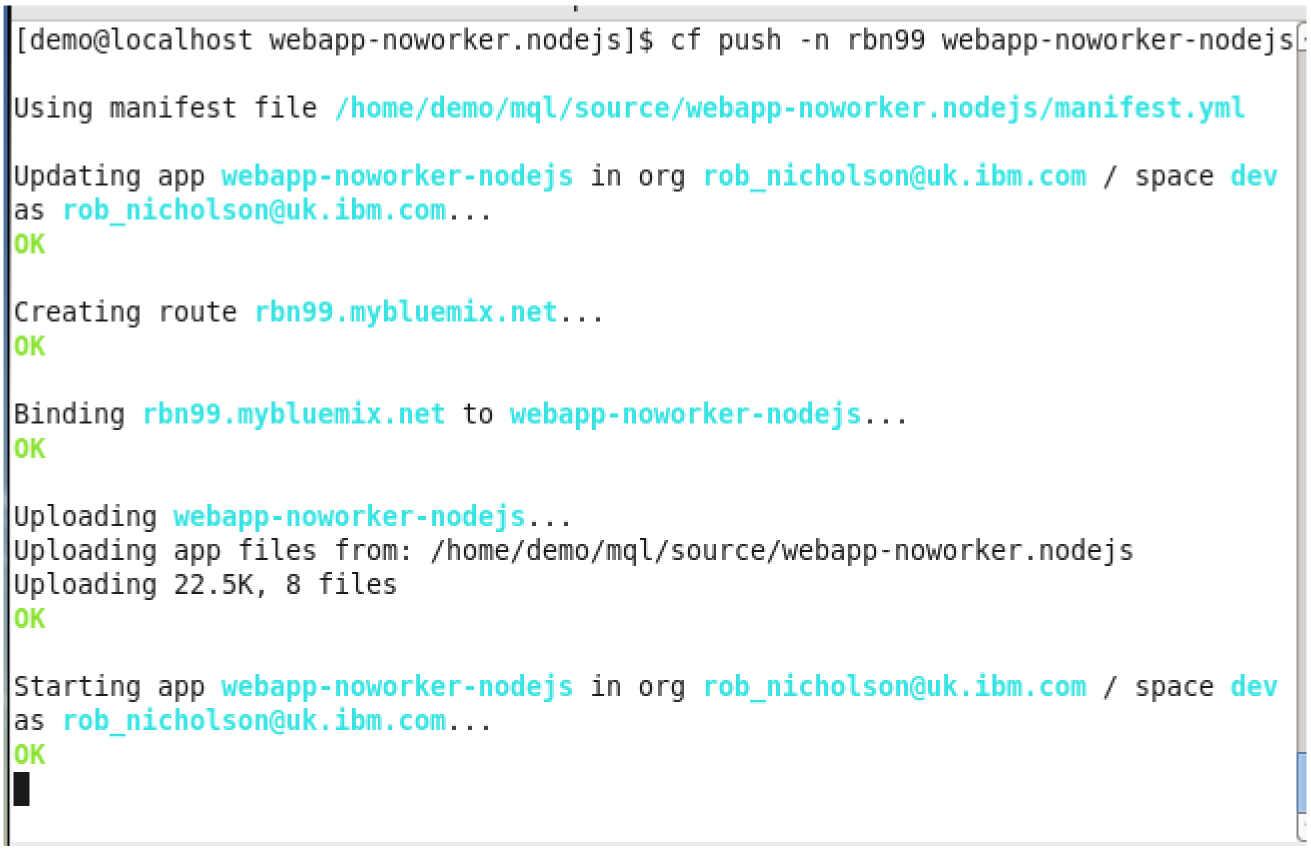
```
[demo@localhost ~]$ cf api https://api.ng.bluemix.net
Setting api endpoint to https://api.ng.bluemix.net...
OK

API endpoint: https://api.ng.bluemix.net (API version: 2.0.0)
Not logged in. Use 'cf login' to log in.
[demo@localhost ~]$ cf login
API endpoint: https://api.ng.bluemix.net

Username> mwhitehead@uk.ibm.com

Password>
Authenticating...
OK

Targeted org mwhitehead@uk.ibm.com

Targeted space dev

API endpoint: https://api.ng.bluemix.net (API version: 2.0.0)
User:         mwhitehead@uk.ibm.com
Org:          mwhitehead@uk.ibm.com
Space:        dev
[demo@localhost ~]$
```

2. In the same terminal window navigate back to the */home/demo/mql/source/webapp-noworker.nodejs* directory we used earlier.

The app.js file should still contain your Twitter API keys from earlier.

3. Push the application to BlueMix by running the following command in the terminal window: *cf push -n <CHOOSE A URL PREFIX HERE> webapp-noworker-nodejs*

You must specify a URL prefix so that when the application is deployed BlueMix can create a unique hostname for it. Choose something like your initials



```
[demo@localhost webapp-noworker.nodejs]$ cf push -n rbn99 webapp-noworker-nodejs

Using manifest file /home/demo/mql/source/webapp-noworker.nodejs/manifest.yml

Updating app webapp-noworker-nodejs in org rob_nicholson@uk.ibm.com / space dev
as rob_nicholson@uk.ibm.com...
OK

Creating route rbn99.mybluemix.net...
OK

Binding rbn99.mybluemix.net to webapp-noworker-nodejs...
OK

Uploading webapp-noworker-nodejs...
Uploading app files from: /home/demo/mql/source/webapp-noworker.nodejs
Uploading 22.5K, 8 files
OK

Starting app webapp-noworker-nodejs in org rob_nicholson@uk.ibm.com / space dev
as rob_nicholson@uk.ibm.com...
OK
```

4. In your browser open a tab and navigate to the URL which BlueMix generated for your application, e.g. http://<your-prefix-here>.ng.bluemix.net/ demo web page which shows the tweets arriving on the left and graph being drawn on the right. You can see the same behaviour we saw when we ran the application locally. Each tweet that arrives has to be processed in the same thread so news tweets can't be shown as they are received.

Congratulations! You've run your first application in BlueMix!

Now let's improve it by using the offloading application and the worker application that we used earlier.

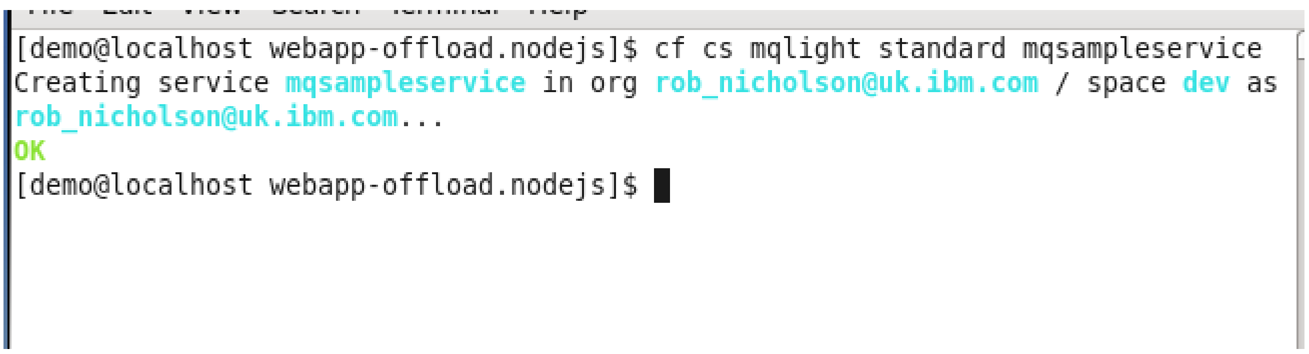
5. Stop the application by running the command "*cf stop webapp-noworker.nodejs*" in the terminal

6. Using the same terminal window navigate to */home/demo/mql/source/webapp-offload.nodejs* and push it to BlueMix using the command *cf push -n <YOUR URL PREFIX HERE> --no-start webapp-offload.nodejs*

Notice how we use the –no-start directive to prevent the app from starting once it's deployed.

7. Create an instance of the MQ Light service that our application can connect to.

Run the command *cf cs mqlight standard mqsampleservice* to create an MQ Light service called mqsampleservice with the "standard" plan.

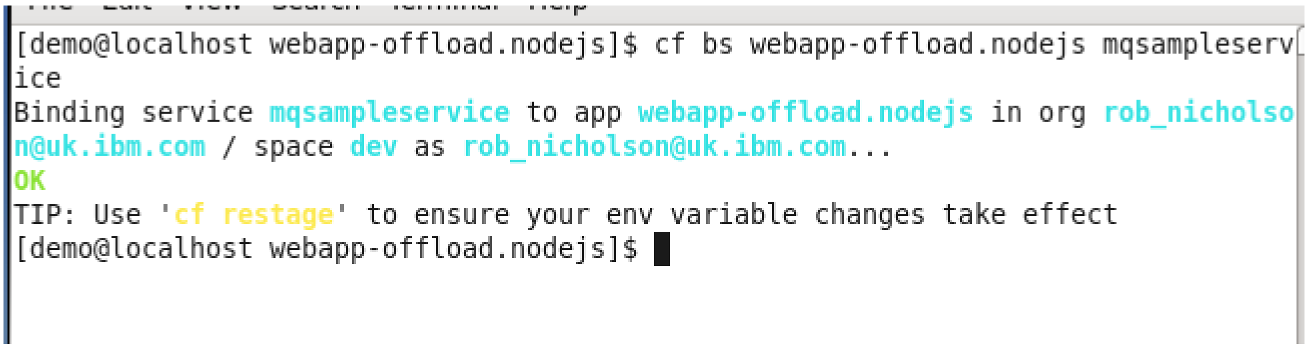


```
[demo@localhost webapp-offload.nodejs]$ cf cs mqlight standard mqsampleservice
Creating service mqsampleservice in org rob_nicholson@uk.ibm.com / space dev as
rob_nicholson@uk.ibm.com...
OK
[demo@localhost webapp-offload.nodejs]$
```

8. Bind the application to the service.

This tells the application which MQ Light service to use. Advanced BlueMix users might have several different MQ Light services which they use for different applications. Even though we only have one service we still need to tell the application which service to bind to.

Run the command *cf bs webapp-offload.nodejs mqsampleservice* to bind the application to the mqsampleservice.



```
[demo@localhost webapp-offload.nodejs]$ cf bs webapp-offload.nodejs mqsampleserv
ice
Binding service mqsampleservice to app webapp-offload.nodejs in org rob_nicholso
n@uk.ibm.com / space dev as rob_nicholson@uk.ibm.com...
OK
TIP: Use 'cf restage' to ensure your env variable changes take effect
[demo@localhost webapp-offload.nodejs]$
```

9. Deploy the worker application and bind it to the same MQ Light service.

Navigate to /home/demo/mql/source/worker.nodejs in the terminal window.

Run the following commands to push the worker application into BlueMix and bind it with the same MQ Light service:

cf push --no-start --no-route worker.nodejs

cf bs worker.nodejs mqsampleservice

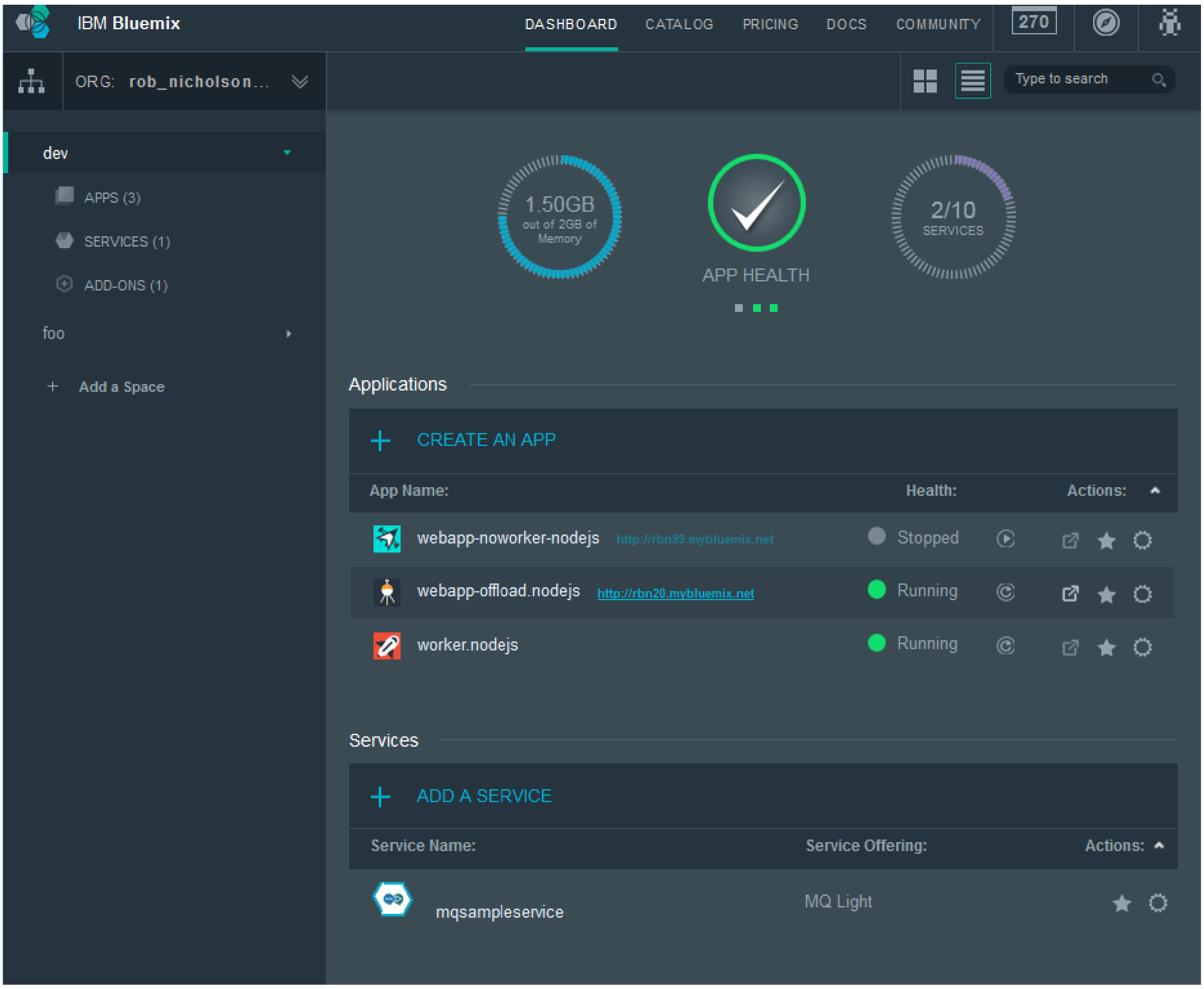
10. Start both of the applications – the offloader and the worker – by running the following commands:

cf start worker.nodejs

cf start webapp-offload.nodejs

11. Examine the MQ Light GUI in Bluemix.
The same GUI that you used in part 2 and 3 of this lab is available in Bluemix. To use it, in a web browser visit. https://ace.ng.bluemix.net/.

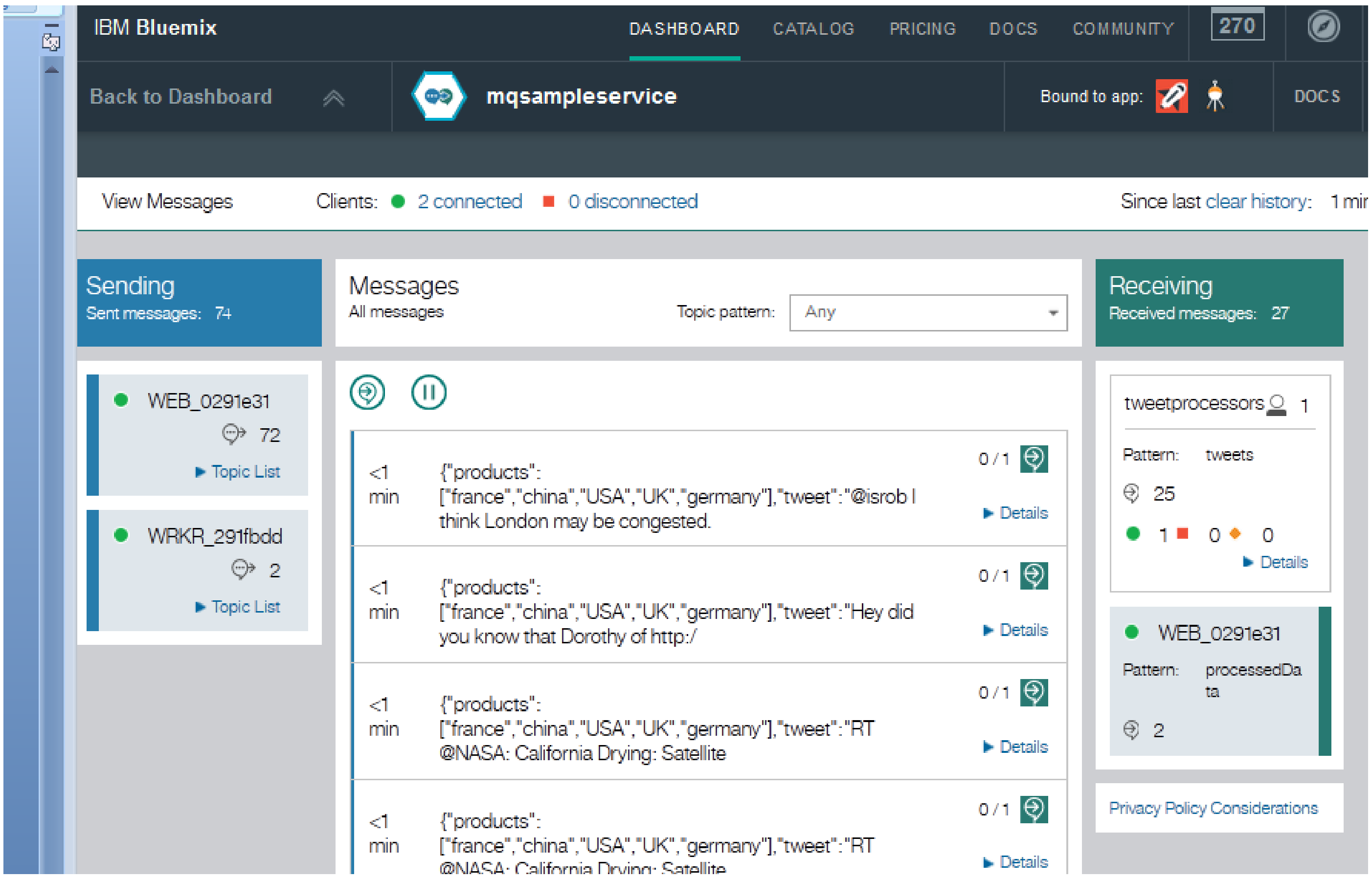
Log in if you are not already logged in and click on the Dashboard link at the top of the screen.



App Name:		Health:	Actions:
webapp-noworker-nodejs	http://rbn39.mybluemix.net	Stopped	
webapp-offload.nodejs	http://rbn20.mybluemix.net	Running	
worker.nodejs		Running	

Service Name:	Service Offering:	Actions:
mqsampleservice	MQ Light	

Under services, you should see the MQ Light service we created on the commandline and called “mqsampleservice”. Click on this to bring up the MQ Light GUI.



```
<1 min  {"products":["france","china","USA","UK","germany"],"tweet":"@isrob I think London may be congested.   0 / 1  Details
<1 min  {"products":["france","china","USA","UK","germany"],"tweet":"Hey did you know that Dorothy of http:/   0 / 1  Details
<1 min  {"products":["france","china","USA","UK","germany"],"tweet":"RT @NASA: California Drying: Satellite   0 / 1  Details
<1 min  {"products":["france","china","USA","UK","germany"],"tweet":"RT @NASA: California Drying: Satellite   0 / 1  Details
```

12. Scale up the number of worker applications

Because BlueMix manages our applications for us we can easily ask it to create more than one instance of them.

To tell BlueMix to create another worker thread application, run the command "*cf scale worker.nodejs -i 2*"

You should be able to see the effect of this in the GUI as another worker connects to the tweetprocessors share.

Part 5 – Changing the worker application from node.js to JMS

So far both the offloading application and the workloader application have been written in node.js using the MQ Light API.

As well as node.js, BlueMix has first class support for deploying and running JMS applications.

The MQ Light service supports MQ Light apps and JMS apps binding to it, allowing you to communicate between applications written using either API.

In the final part of the lab we will change the worker application from an MQ Light application running in node.js to a JMS application running in Java. The offloading application will remain the same. All we're changing is the worker application.

1. Stop the two applications that we currently have running in BlueMix

cf stop worker.nodejs

cf stop webapp-offload.nodejs

You can check that no apps are running by using the command:

cf apps

2. Deploy the JMS application

In the terminal window navigate to */home/demo/mql/source/worker.JMS*

Using the same commands that we used earlier, push the JMS worker to BlueMix and bind it to the mqsampleservice:

cf push --no-start --no-route worker.JMS

cf bs worker.JMS mqsampleservice

The offloading application is still running so all we need to do start the offloader and the JMS worker:

cf start worker.JMS

cf start webapp-offload.nodejs

3. Check that the applications are running correctly

Use the "*cf apps*" command to check that your applications are running correctly.

In your browser navigate back to the demo web page and check that the Twitter feed is being updated as soon as tweets arrive, and that the graph is updating correctly. Remember we now only have 1 instance of the JMS worker application – we need to tell BlueMix to scale up the number of worker applications again.

Run the same command that we used earlier to scale up the number of JMS workers:

cf scale worker.JMS -i 2

Check that the graph on the web page is updating more quickly as the worker threads cope with the workload more easily.

Congratulations! You've reached the end of the lab!

We've shown you how to use MQ Light to rapidly develop and debug your applications and you've learnt how MQ Light can make your applications more responsive.

You've also seen how the MQ Light API allows you to create multiple workers by joining them into a share, allowing you to build a scalable app that can respond to change in demand.

Finally you've seen how you can deploy your applications to BlueMix, leaving you to worry about coding your applications while BlueMix manages the infrastructure and scales your application quickly and easily.

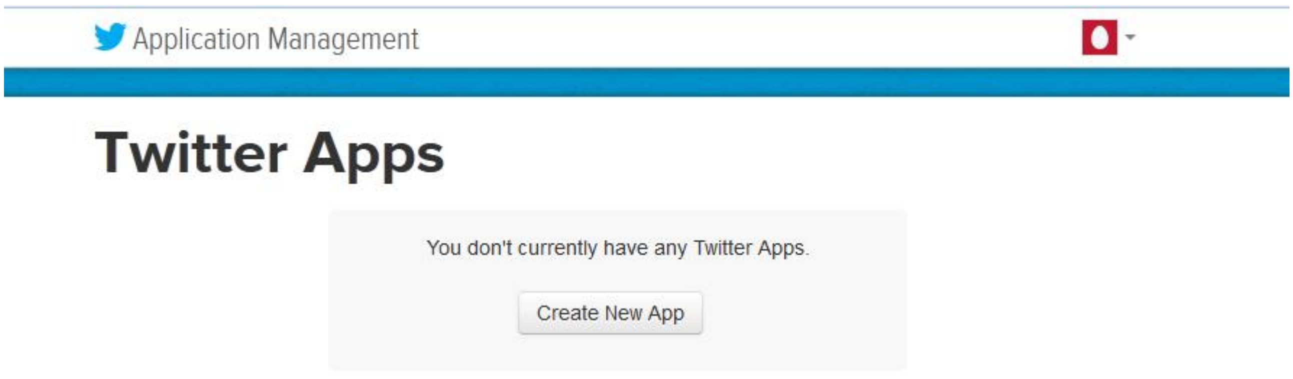
Appendix: Obtaining your own twitter credentials.

The sample applications used in this lab use twitter as a data source.

Twitter requires us to authenticate with a set of temporary developer credentials. Before you can run the sample application you must have a set of credentials to use. For convenience during the scheduled lab sessions, a set of credentials are provided. These credentials will be deactivated at the end of the scheduled lab session. If you wish to repeat the lab later you will need to generate your own credentials. This appendix describes how to generate your own set of twitter credentials.

1. Sign in to Twitter

In your browser navigate to https://apps.twitter.com/ and sign-in at the top right of the screen (you will need a Twitter ID).



2. Create an application by pressing the 'Create New App' button

Once you are logged in you should see a button called 'Create New App' which you can click on to start creating your first application.

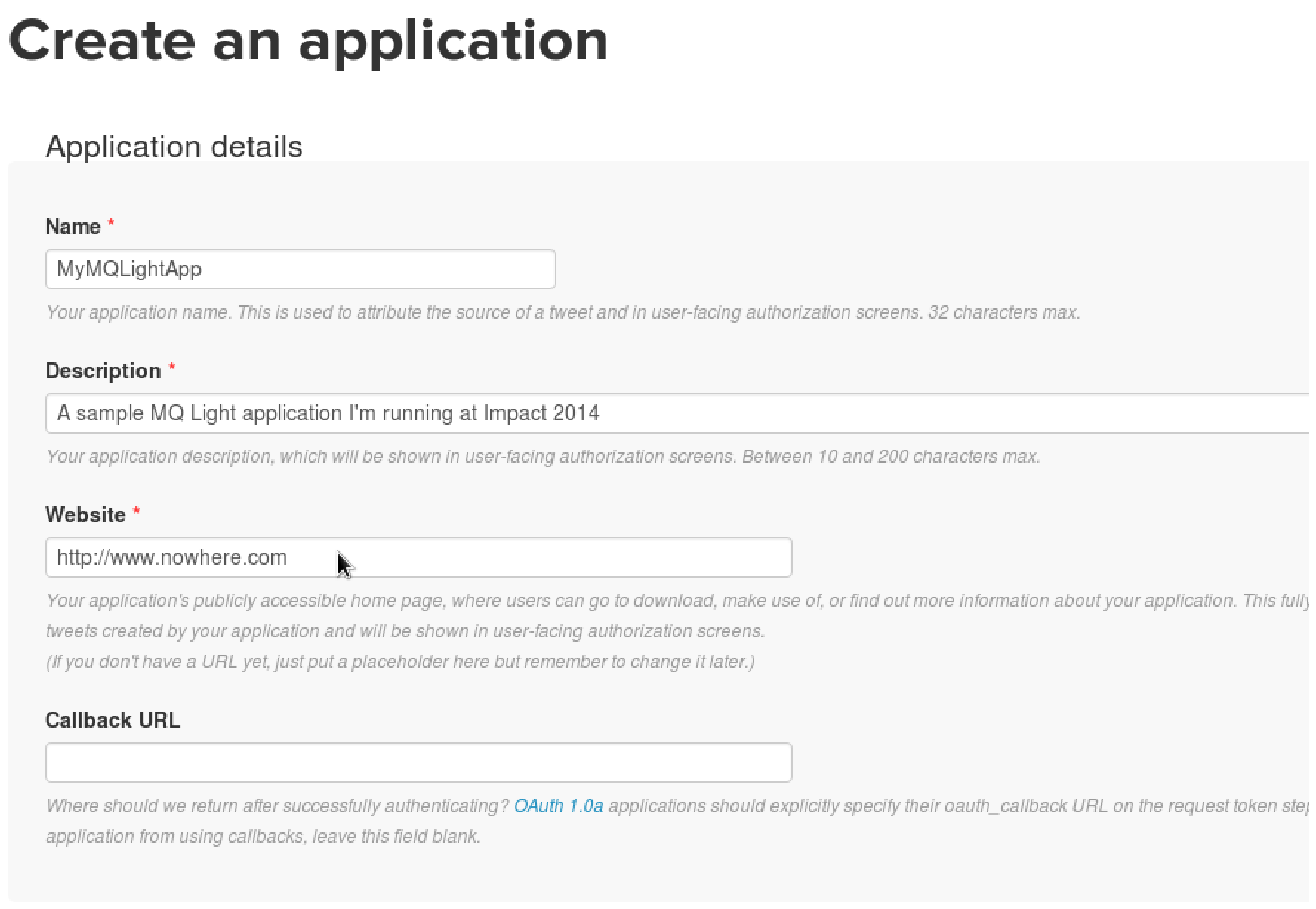
3. Fill out the fields in the form

The name for your application must be unique so pick a name that is unlikely to have been used already.

The description text does not matter.

The website URL does not need to be a valid URL.

You can leave the callback URL blank.



Create an application

Application details

Name *

MyMQLightApp

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

A sample MQ Light application I'm running at Impact 2014

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

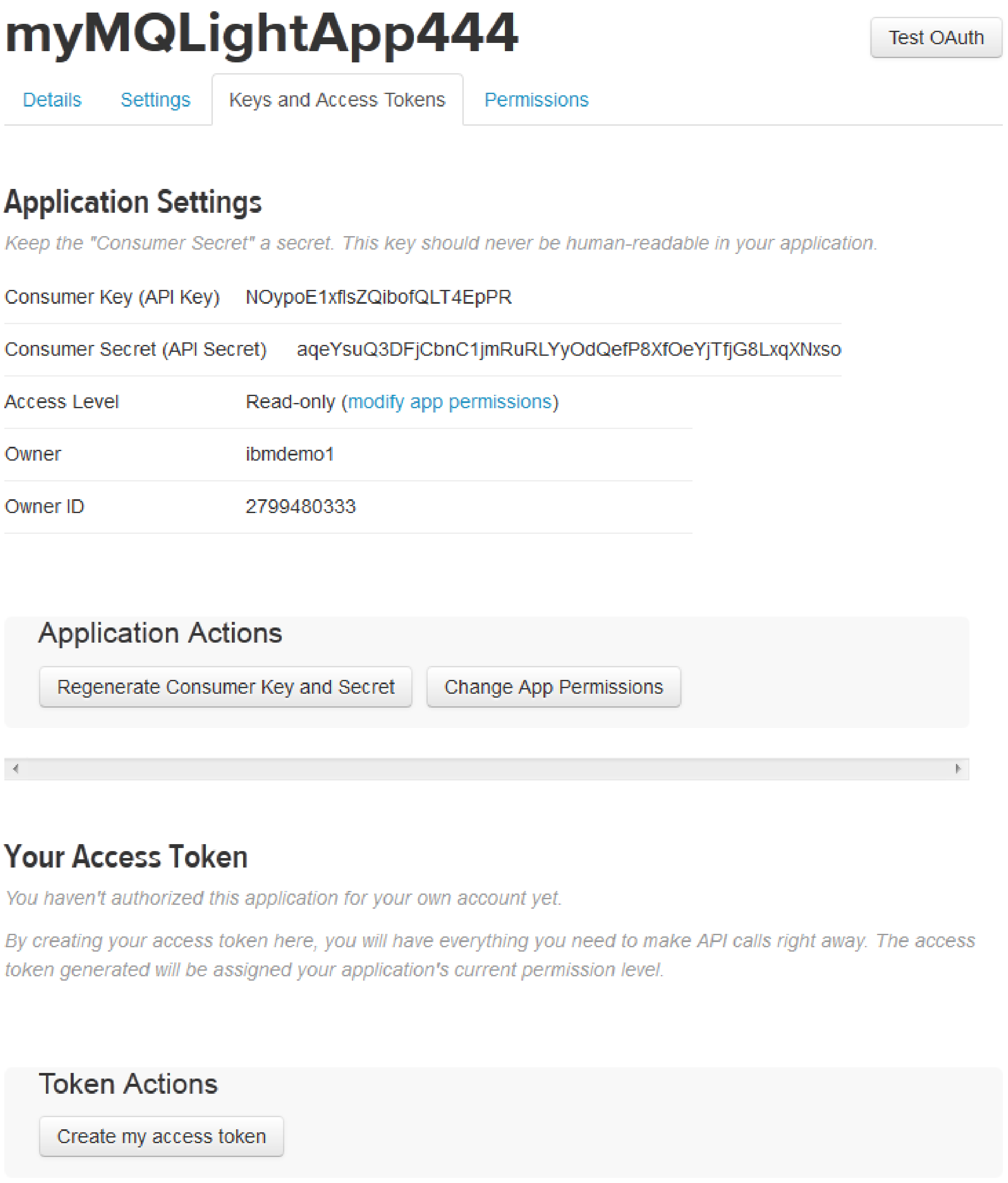
http://www.nowhere.com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step application from using callbacks, leave this field blank.

4. Once you have created your application, go to the “Keys and Access Tokens” tab and scroll to the bottom of the page. Select the option to “Create my access token”:


myMQLightApp444

Test OAuth

Details | Settings | Keys and Access Tokens | Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	NOypoE1xflsZQibofQLT4EpPR
Consumer Secret (API Secret)	aqeYsuQ3DFjCbnC1jmRuRLYyOdQefP8XfOeYjTfjG8LxqXNxso
Access Level	Read-only (modify app permissions)
Owner	ibmdemo1
Owner ID	2799480333

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

You haven't authorized this application for your own account yet.

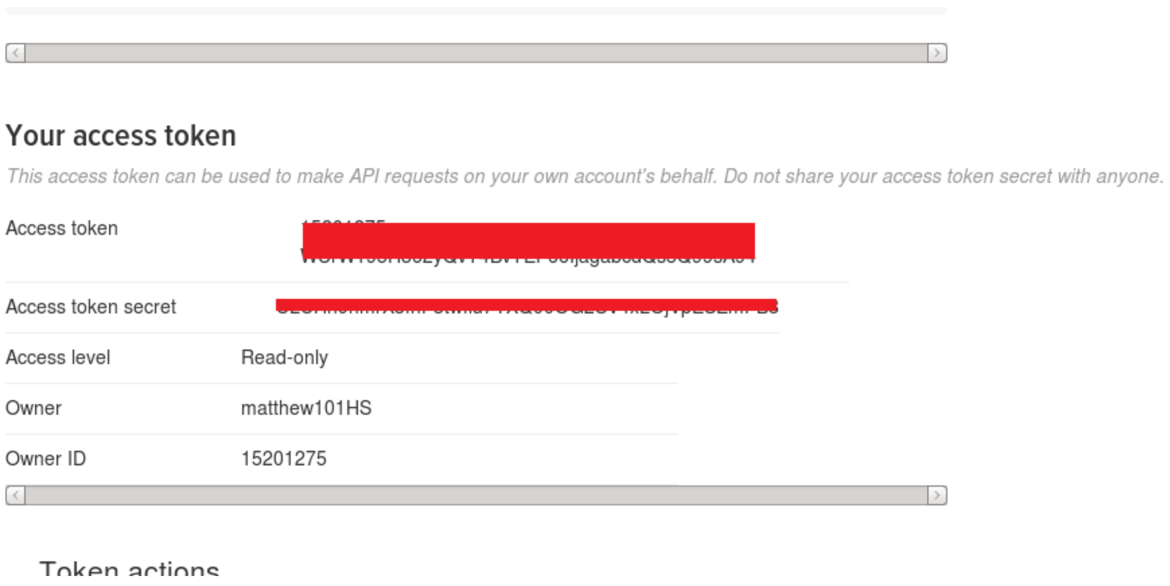
By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

Create my access token

5. It may take a moment for the keys to be generated. There is a 'refresh' option at the top of the page which you can use if the keys take a while to generate.

Once they have been generated you can minimise the browser and we'll come back to retrieve them later.


Your access token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

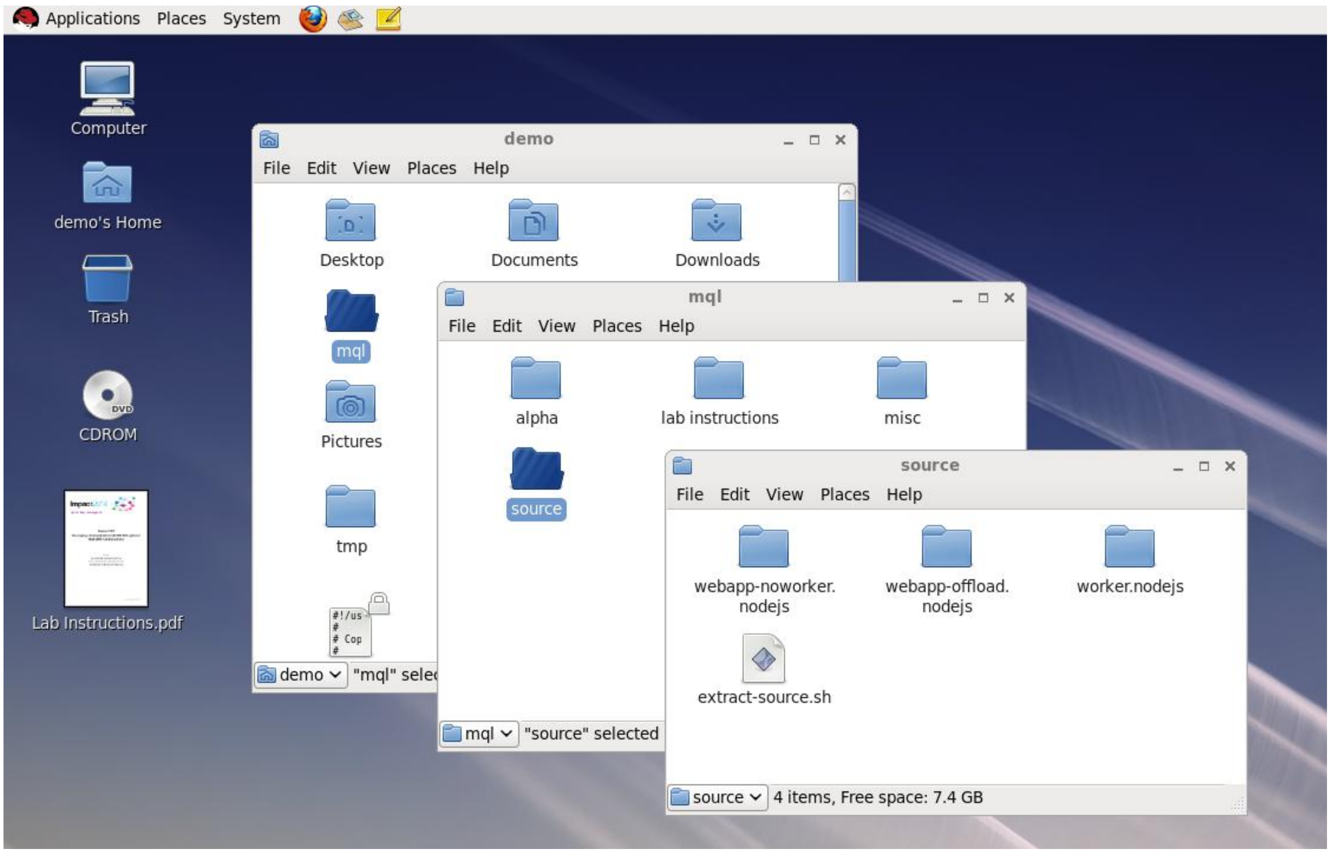
Access token	[redacted]
Access token secret	[redacted]
Access level	Read-only
Owner	matthew101HS
Owner ID	15201275

Token actions

6. Open the sample application and locate the sample files that we need to insert the tokens you generated into.

All of the files for this lab are inside the ***demo's Home*** folder under a directory called *mql*.

From the desktop open ***demo's Home*** folder and navigate into the *mql* and then into *mql/source*.



The file we need to change is twitterkey.json. There is a version of this file in each of the application folders inside */home/demo/mql/source*

Open each of the folders in turn and open *twitterkey.json* in the text editor (Note: don't double click the files to edit it, instead you need to right-click on each file and select the “Open with Other Application” option. Choose gedit to open the files with)

The file should look like this:

```
{
    "consumer_key": "XXXXXXXXXXXXXXXXXX",
    "consumer_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "access_token_key": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "access_token_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

7. Replace the key and secret strings in the files with the values in your browser that you minimised earlier.

Set *consumer_key* in the file to the API Key string in your browser
Set *consumer_secret* in the file to the API Secret string in your browser
Set *access_token_key* in the file to the Access Token string in your browser
Set *access_token_secret* in the file to the Access Token Secret string in your browser

Save the files and close the text editors.

8. Congratulations you are now ready to continue the lab using twitter keys belonging to an account that you own. Return to Part 1.