```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Reactive.Linq;
5  using System.Text;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Newtonsoft.Json;
9  using RabbitMQ.Client;
10 using RabbitMQ.Client.Framing;
11 using RabbitRx.Advanced.Subscription;
12 using RabbitRx.Core.Message;
13 using RabbitRx.Core.Subscription;
14 using RabbitRx.Json.Subscription;
15 using System.IO;
16
17 namespace Sender
18 {
19     class Program
20     {
21         /// <summary>
22         /// Connecting to a Broker
23         /// </summary>
24         static readonly ConnectionFactory factory = new ConnectionFactory
             { HostName = "66.128.60.46", UserName = "dev", Password = "dev",
             VirtualHost = "/" };
25         static readonly IConnection connection = factory.CreateConnection();
26         static readonly IModel channel = connection.CreateModel();
27         static string exchangeName = "deviceTopic";
28         static string syntacticAnalyzerQueue = "syntacticAnalyzer";
29         static string discoveryResponseQueue = "discoveryResponse";
30         static string authenticationResponseQueue = "authenticationResponse";
31
32         static List<Model.DataTime> listTime = new List<Model.DataTime>();
33
34         static void Main(string[] args)
35         {
36             channel.ExchangeDeclare(exchangeName, "topic");
37             //Queue to send data
38             channel.QueueDeclare(syntacticAnalyzerQueue, false, false, false,
                 null);
39             channel.QueueBind(syntacticAnalyzerQueue, exchangeName,
                 syntacticAnalyzerQueue);
40
41             Start();
42         }
43
44         private static CancellationTokenSource tokenSource;
45         private static CancellationTokenSource tokenSourceDiscovery;
46         private static CancellationTokenSource tokenSourceAuthentication;
47         private static CancellationTokenSource tokenSourceProducer;
48         private static StreamWriter csv;
```

```
49          private static string csvName;
50
51          /// <summary>
52          /// Title: RabbitRx
53          /// Author: Ben Johnson
54          /// Date: Jan 27, 2015
55          /// Availability: https://github.com/bensmind/RabbitRx
56          /// </summary>
57          private static void Start()
58          {
59              tokenSource = new CancellationTokenSource();
60              tokenSourceDiscovery = new CancellationTokenSource();
61              tokenSourceAuthentication = new CancellationTokenSource();
62              tokenSourceProducer = new CancellationTokenSource();
63
64              Console.WriteLine("Enter the number of devices:");
65              var devicesCount = Console.ReadLine();
66              Console.WriteLine("\nEnter Lambda:");
67              var lambda = Console.ReadLine();
68              Console.WriteLine("\nEnter the full path:");
69              csvName = Console.ReadLine();
70              Console.WriteLine("\nSender: Press Enter to Start");
71              Console.ReadLine();
72              if(!string.IsNullOrEmpty(csvName))
73                  csv = new StreamWriter(csvName);
74              Task.Run(() => Produce(int.Parse(devicesCount), double.Parse
                  (lambda)));
75              Task.Run(() => ConsumeThrottleDiscovery());
76              Task.Run(() => ConsumeThrottleAuthorization(double.Parse(lambda)));
77              Console.WriteLine("Press Any Key to Stop");
78              Console.ReadLine();
79              tokenSource.Cancel();
80              tokenSourceDiscovery.Cancel();
81              tokenSourceAuthentication.Cancel();
82              tokenSourceProducer.Cancel();
83              Start();
84          }
85
86          private static void Produce(int devicesCount, double lambda)
87          {
88              var deviceCounter = 0;
89              var rand = new Random();
90              var settings = new BasicProperties()
91              {
92                  ContentType = "application/json",
93                  DeliveryMode = 1 //1)not durable, 2)durable
94              };
95
96              var ob = Observable.Generate(Guid.NewGuid(), i => !
                  tokenSource.IsCancellationRequested, i => Guid.NewGuid(), i => i, x
                   => TimeSpan.FromMilliseconds(5)/*TimeSpan.FromMilliseconds
                  (nextTime(1 / lambda))*/);
```

```
 97
 98              ob.Subscribe(id =>
 99              {
100                  deviceCounter++;
101
102                  //Send Data
103                  var device = new Model.Device()
104                  {
105                      Id = id,
106                      Name = "Device #" + deviceCounter,
107                      Type = Enum.DeviceType.Temperature
108                  };
109                  var bytes = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject
                       (device));
110                  channel.BasicPublish(exchangeName, syntacticAnalyzerQueue,
                       settings, bytes);
111                  Console.WriteLine("Published Discovery: {0}", device.Name);
112
113                  //Stop to device registration
114                  if (deviceCounter == devicesCount)
115                  {
116                      tokenSource.Cancel();
117                  }
118
119              }, tokenSource.Token);
120
121          }
122
123          static void ConsumeThrottleDiscovery()
124          {
125              var channelConsumer = connection.CreateModel();
126
127              channelConsumer.BasicQos(0, 50, false);
128              channelConsumer.ExchangeDeclare(exchangeName, "topic");
129              //Declare the Discovery Response Queue
130              channelConsumer.QueueDeclare(discoveryResponseQueue, false, false,
                   false, null);
131              channelConsumer.QueueBind(discoveryResponseQueue, exchangeName,
                   discoveryResponseQueue);
132              //Queue to send data to Syntactic AnalyzerQueue
133              channelConsumer.QueueDeclare(syntacticAnalyzerQueue, false, false,
                   false, null);
134              channelConsumer.QueueBind(syntacticAnalyzerQueue, exchangeName,
                   syntacticAnalyzerQueue);
135
136              var settings = new BasicProperties()
137              {
138                  ContentType = "application/json",
139                  DeliveryMode = 1 //1)not durable, 2)durable
140              };
141
142              //Consumer Discovery
```

```
143             var consumer = new JsonObservableSubscription<object>
                  (channelConsumer, discoveryResponseQueue, true);
144
145             var throttlingConsumer = new
                  ThrottlingConsumer<RabbitMessage<object>>(consumer, 4);
146
147             throttlingConsumer.Subscribe(message =>
148             {
149                 var discoveryResponse =
                      JsonConvert.DeserializeObject<Model.DiscoveryResponse>
                      (message.Payload.ToString());
150                 var ramdomString = Model.Common.RandomString(10);
151                 var password = Model.Common.Encrypt(ramdomString +
                      discoveryResponse.Salt);
152
153                 var authentication = new Model.Authentication()
154                 {
155                     Id = discoveryResponse.Id,
156                     BaseString = ramdomString,
157                     Password = password
158                 };
159                 var bytes = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject
                      (authentication));
160                 channelConsumer.BasicPublish(exchangeName,
                      syntacticAnalyzerQueue, settings, bytes);
161                 Console.WriteLine("Received (Thread {1}): {0}", message.Payload,
                      Thread.CurrentThread.GetHashCode());
162
163             }, tokenSourceDiscovery.Token);
164
165             var start = throttlingConsumer.Start(tokenSourceDiscovery.Token,
                  TimeSpan.FromSeconds(10));
166
167             start.ContinueWith(t =>
168             {
169                 consumer.Close();
170                 channelConsumer.Dispose();
171             });
172         }
173
174         static void ConsumeThrottleAuthentication(double lambda)
175         {
176             var channelConsumer = connection.CreateModel();
177
178             channelConsumer.BasicQos(0, 50, false);
179             channelConsumer.ExchangeDeclare(exchangeName, "topic");
180             //Declare the Authentication Response Queue
181             channelConsumer.QueueDeclare(authenticationResponseQueue, false,
                  false, false, null);
182             channelConsumer.QueueBind(authenticationResponseQueue, exchangeName,
                  authenticationResponseQueue);
183             //Queue to send data to Syntactic AnalyzerQueue
```

```
184             channelConsumer.QueueDeclare(syntacticAnalyzerQueue, false, false,
                  false, null);
185             channelConsumer.QueueBind(syntacticAnalyzerQueue, exchangeName,
                  syntacticAnalyzerQueue);
186
187             var settings = new BasicProperties()
188             {
189                 ContentType = "application/json",
190                 DeliveryMode = 1
191             };
192
193             //Consumer Authentication
194             var consumer = new JsonObservableSubscription<object>
                  (channelConsumer, authenticationResponseQueue, true);
195
196             var throttlingConsumer = new
                  ThrottlingConsumer<RabbitMessage<object>>(consumer, 4);
197
198             throttlingConsumer.Subscribe(message =>
199             {
200                 var authenticationResponse =
                      JsonConvert.DeserializeObject<Model.AuthenticationResponse>
                      (message.Payload.ToString());
201
202                 Task.Run(() => ProduceData(authenticationResponse, lambda));
203                 Console.WriteLine("Received (Thread {1}): {0}", message.Payload,
                      Thread.CurrentThread.GetHashCode());
204
205             }, tokenSourceAuthentication.Token);
206
207             var start = throttlingConsumer.Start(tokenSourceAuthentication.Token,
                    TimeSpan.FromSeconds(10));
208
209             start.ContinueWith(t =>
210             {
211                 consumer.Close();
212                 channelConsumer.Dispose();
213             });
214         }
215
216         static void ProduceData(Model.AuthenticationResponse auth, double lambda)
217         {
218             var packageCounter = 0;
219             var rand = new Random();
220             var settings = new BasicProperties()
221             {
222                 ContentType = "application/json",
223                 DeliveryMode = 1
224             };
225
226             var ob = Observable.Generate(rand.Next(), i => !
                  tokenSourceProducer.IsCancellationRequested, i => rand.Next(), i =>
```

```
                i, x => TimeSpan.FromMilliseconds(nextTime(1 / lambda)));
227
228             ob.Subscribe(id =>
229             {
230                 var date = DateTime.Now;
231                 if (!activeBackPressure(auth.Id, date))
232                 {
233                     packageCounter++;
234
235                     //Send Data
236                     var data = new Model.Data()
237                     {
238                         Id = auth.Id,
239                         IdTransaction = packageCounter,
240                         Token = auth.Token,
241                         Value = rand.Next()
242                     };
243                     var jsonData = JsonConvert.SerializeObject(data);
244                     var bytes = Encoding.UTF8.GetBytes(jsonData);
245                     channel.BasicPublish(exchangeName,"dataManager", settings,
                         bytes);
246                     Console.WriteLine("Published Data: {0}", jsonData);
247                     if (!string.IsNullOrEmpty(csvName))
248                     {
249                         var newLine = string.Format("{0},{1},{2}", auth.Id,
                         packageCounter, date.ToString("MM/dd/yyyy hh:mm:ss.fff tt"));
250                         csv.WriteLine(newLine);
251                         csv.Flush();
252                     }
253                 }
254
255             }, tokenSourceProducer.Token);
256         }
257
258         private static bool activeBackPressure(Guid idDevice, DateTime date)
259         {
260             if (listTime.Count == 0)
261             {
262                 listTime.Add(new Model.DataTime() { date = date, idDevice =
                    idDevice });
263                 return false;
264             }
265             else
266             {
267                 foreach (var dataTime in listTime)
268                 {
269                     if(dataTime.idDevice == idDevice)
270                     {
271                         if(dataTime.date.Hour == date.Hour &&
                         dataTime.date.Minute == date.Minute && dataTime.date.Second
                         == date.Second)
272                         {
```

```
273                             return true;
274                         }
275                         else
276                         {
277                             dataTime.date = date;
278                             return false;
279                         }
280                     }
281                 }
282             }
283             listTime.Add(new Model.DataTime() { date = date, idDevice =
                  idDevice });
284             return false;
285         }
286 
287         private static double nextTime(double rateParameter)
288         {
289             Random random = new Random();
290             return -Math.Log(1.0 - random.NextDouble()) / rateParameter;
291         }
292     }
293 }
294 
```