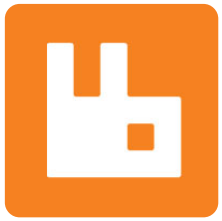


Cómo utilizar colas de mensajes con RabbitMQ - Parte II

PUBLICADO POR: JONATHAN WIESEL (/AUTHOR/JONATHAN.HTML), EL 11/03/2014

Detalles del Curso:

Dificultad: Aprendiz
Duración: 20 min

Hemos habilitado un repo en GitHub para que puedas descargar el código de esta entrada:
Échale un vistazo aquí. (https://github.com/codeheroco/rabbitmq)

La semana pasada (http://codehero.co/como-utilizar-colas-de-mensajes-con-rabbitmq-parte/) comenzamos a ver los conceptos básicos de las colas de mensajes y las ventajas que nos podría traer al utilizar RabbitMQ como nuestra solución, esta vez nos adentraremos en el funcionamiento de RabbitMQ al colocarlo en acción y veremos un poco sobre su interfaz de administración.

Administración

Esta herramienta nos permitirá realizar una gestión completa de la mayoría de los procesos que están involucrados en las colas de mensajes. Para ello debemos habilitar el *plugin* y luego iniciar el servicio de RabbitMQ:

Debian

```
$ sudo rabbitmq-plugins enable rabbitmq_management
$ sudo service rabbitmq-server start
```

Windows

Nos dirigiremos al directorio de instalación de RabbitMQ y luego al subdirectorio `sbin` del mismo, algo asi:

```
%PROGRAMFILES%\RabbitMQ Server\rabbitmq_server_2.7.1\sbin\
```

Y luego ejecutamos el siguiente comando:

```
$ rabbitmq-plugins.bat enable rabbitmq_management
```

Debemos reinstalar el servicio de de RabbitMQ para que el plugin funcione por lo que haremos lo siguiente:

```
$ rabbitmq-service.bat stop
$ rabbitmq-service.bat install
$ rabbitmq-service.bat start
```

Mac OS X

> Si instalaste RabbitMQ con Homebrew (http://codehero.co/como-lo-hago-instalar-homebrew/) este plugin viene habilitado por defecto.

Iniciemos el servicio de RabbitMQ:

```
$ brew services start rabbitmq
```

Prueba de administración

Finalmente probaremos que el servicio está ejecutandose ejecutando:

```
$ rabbitmqctl status
```

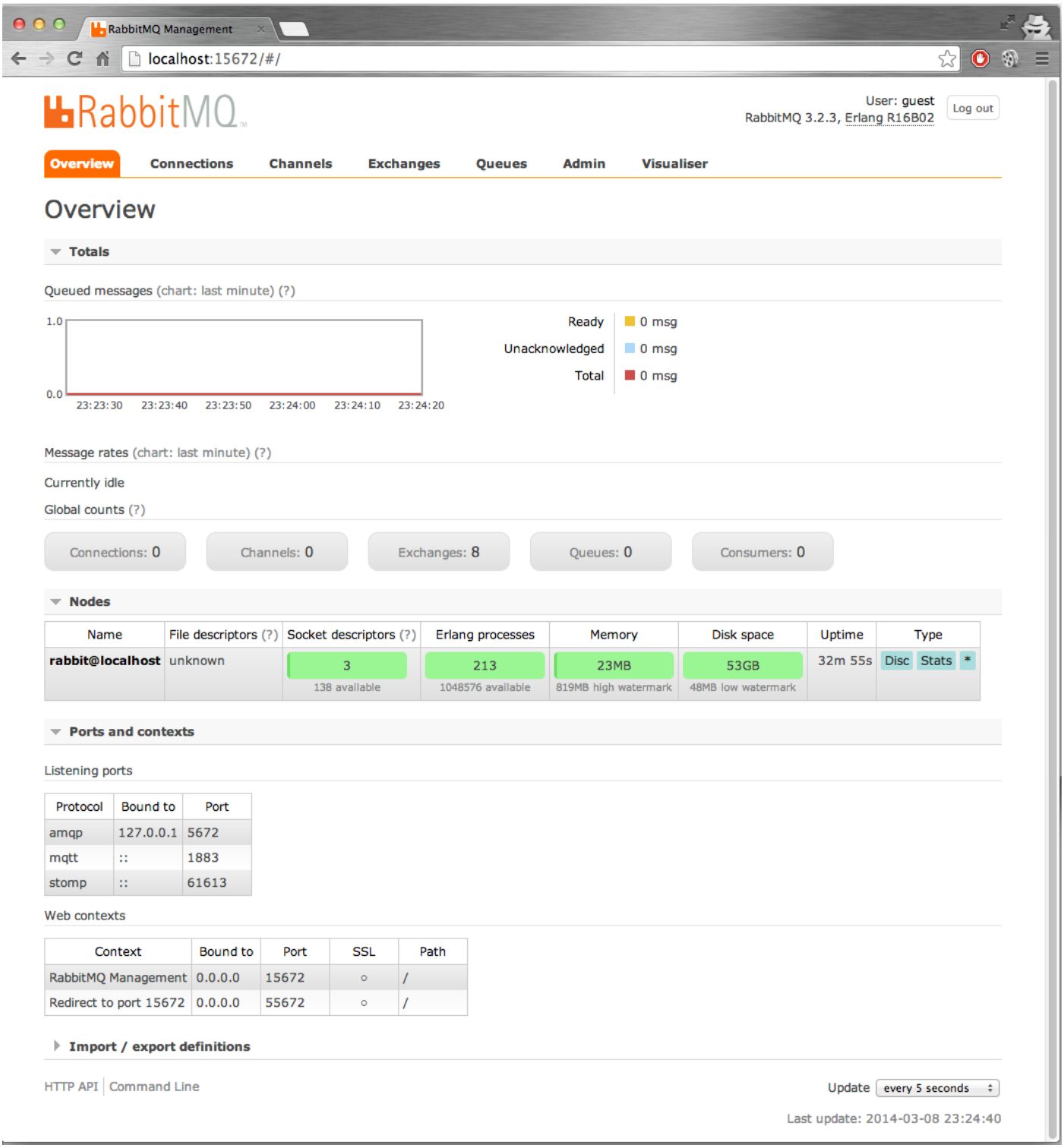
Y obtendremos una salida de gran parte del estado y configuración del servidor de RabbitMQ como esta:

```
[{pid,10062},
 {running_applications,
     [{rabbitmq_management_visualiser,"RabbitMQ Visualiser","3.2.3"},
      {rabbitmq_management,"RabbitMQ Management Console","3.2.3"},
      {rabbitmq_web_dispatch,"RabbitMQ Web Dispatcher","3.2.3"},
      {webmachine,"webmachine","1.10.3-rmq3.2.3-gite9359c7"},
      {mochiweb,"MochiMedia Web Server","2.7.0-rmq3.2.3-git680dba8"},
      {rabbitmq_mqtt,"RabbitMQ MQTT Adapter","3.2.3"},
      {rabbitmq_stomp,"Embedded Rabbit Stomp Adapter","3.2.3"},
      {rabbitmq_management_agent,"RabbitMQ Management Agent","3.2.3"},
      {rabbitmq_amqp1_0,"AMQP 1.0 support for RabbitMQ","3.2.3"},
      {rabbit,"RabbitMQ","3.2.3"},
      {os_mon,"CPO  CXC 138 46","2.2.13"},
      {inets,"INETS  CXC 138 49","5.9.6"},
      {mnesia,"MNESIA  CXC 138 12","4.10"},
      {amqp_client,"RabbitMQ AMQP Client","3.2.3"},
      {xmerl,"XML parser","1.3.4"},
      {sasl,"SASL  CXC 138 11","2.3.3"},
      {stdlib,"ERTS  CXC 138 10","1.19.3"},
      {kernel,"ERTS  CXC 138 10","2.16.3"}]},
 {os,{unix,darwin}},
 {erlang_version,
     "Erlang R16B02 (erts-5.10.3) [source] [smp:2:2] [async-threads:30] [hipe] [
 {memory,
     [{total,24239512},
      {connection_procs,2888},
      {queue_procs,2888},
      {plugins,149124},
      {other_proc,9095428},
      {mnesia,32072},
      {mgmt_db,47388},
      {msg_index,12180},
      {other_ets,641380},
      {binary,13080},
      {code,10763298},
      {atom,531937},
      {other_system,2947849}]},
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,858993459},
 {disk_free_limit,50000000},
```

```
41    {disk_free,57325109248},
42    {file_descriptors,
43        [{total_limit,156},{total_used,5},{sockets_limit,138},{sockets_used,3}]},
44    {processes,[{limit,1048576},{used,209}]},
45    {run_queue,0},
46    {uptime,9}]
47   ...done.
```

Probemos la interfaz gráfica ingresando a nuestro navegador y dirigiendonos a `http://localhost:15672/`, este es el puerto por defecto del submódulo de administración de RabbitMQ:

Se nos presentará una pantalla de inicio de sesión (seguridad ante todo). Por ser una instalación nueva, el usuario y clave por defecto es `guest` para ambos.
Dentro podremos ver muchos detalles acerca del estado de nuestro sistema de colas, entre ellos podremos apreciar datos importantes como las conexiones existentes, las colas definidas, los intercambiadores, consumidores de mensajes, gestión administrativa del servidor y mucho más.



No te preocupes si no entiendes mucho de lo que hay aquí, más adelante cuando nos pongamos en acción con algunos ejemplos podrás apreciarlo mejor.

Demo

Bien, es hora de probar cómo funciona todo esto, que hemos visto.
Para nuestro ejemplo utilizaremos node.js (), el novedoso framework de javascript, esto con la finalidad de motivarlos a introducirse en el uso de este lenguaje y por su facilidad de lectura y aprendizaje.

> No te preocupes, RabbitMQ tiene librerías para casi todos los lenguajes (http://www.rabbitmq.com/devtools.html) para que puedas desarrollar en el lenguaje de tu preferencia.
> El código que verás aquí se encuentra en el repositorio de Github referenciado al comienzo de esta entrada. En él se encuentran las instrucciones para puedas correr tu mismo los programas y puedas ver en vivo lo que sucede.

Veamos primero el código de nuestro **publicador** o cliente generador de mensajes:

```
1  var amqp = require('amqp');
2  var helper = require('./amqp-hacks');
3
4  var conexion = amqp.createConnection({host: 'localhost'});
5
6  conexion.on('ready', function(){
7
8      var mensaje = 'Hola CODEHERO ' + new Date();
9
10     conexion.publish('sencilla', mensaje);
11
12     helper.safeEndConnection(conexion);
13 });
```

Detallemos paso a paso lo que hace cada línea:

- Requerimos el módulo del protocolo AMQP que usaremos para comunicarnos con RabbitMQ.
- Requerimos un *script* local que nos ayudará a finalizar correctamente la conexión.
- Establecemos la conexión con nuestro servidor de RabbitMQ.
- Cuando la conexión está establecida proseguimos con lo siguiente:
  - Construimos el mensaje.
  - Enviamos el mensaje indicando la cola `sencilla` a la cual debemos enviarlo.
  - Finalizamos la conexión.

Bastante sencillo ¿no lo crees?
Observemos ahora el código de nuestro **consumidor** o servidor receptor:

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

var conexion = amqp.createConnection({host: 'localhost'});

conexion.on('ready', function(){
    conexion.queue('sencilla', {autoDelete: false}, function(cola){
        cola.subscribe(function(mensaje){
            console.log('Mensaje recibido -> %s', mensaje.data.toString('utf-8')
        });
    });
});
```

Nuevamente detallando cada línea:

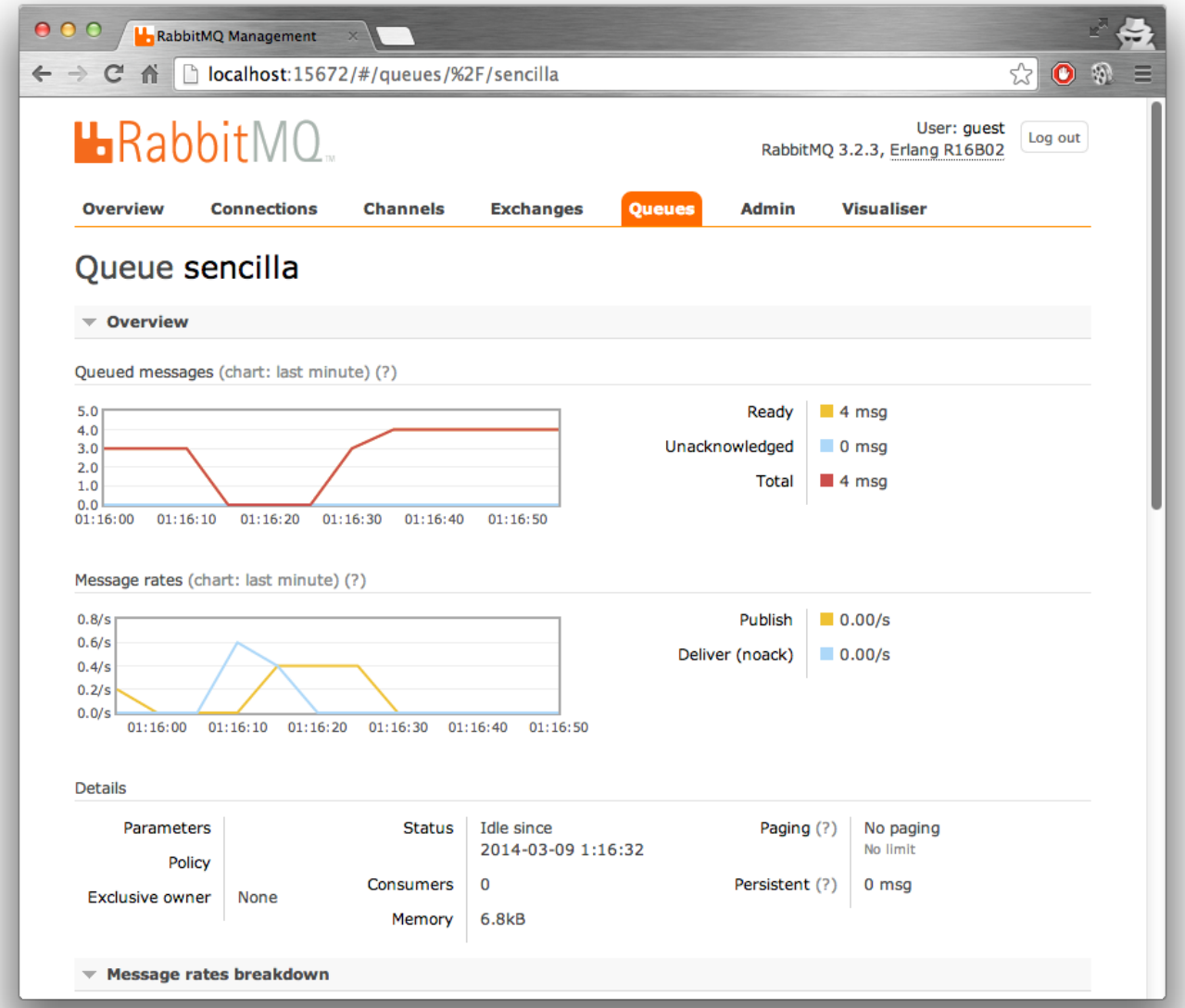
- Requerimos el módulo del protocolo AMQP.
- Establecemos la conexión con nuestro servidor de RabbitMQ.
- Cuando la conexión está establecida proseguimos con lo siguiente:
  - Instanciamos la cola `sencilla` indicando la opción para que esta no sea eliminada cuando no existan más mensajes en ella.
    - Nos suscribimos a la cola (esto producirá que el programa se quede oyendo a la cola).
      - Imprimimos el mensaje cada vez que la cola recibe y nos envía.

Podemos probar iniciando nuestro **consumidor** e ir enviado mensajes con nuestro **publicador** para observar en la salida como es el flujo de mensajes el cual terminaría siendo algo así:

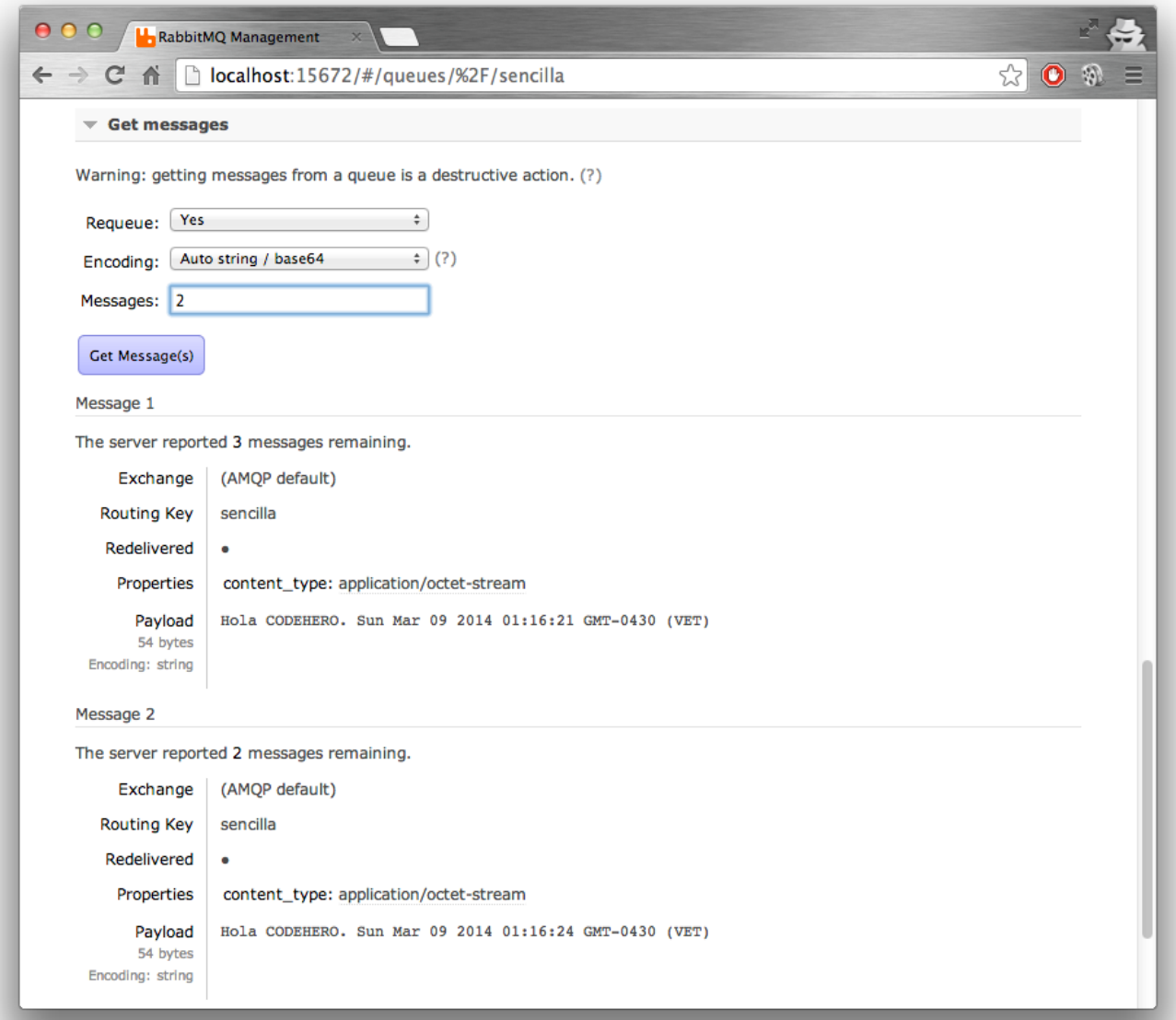
```
Mensaje recibido -> Hola CODEHERO. Sun Mar 09 2014 01:05:30 GMT-0430 (VET)
Mensaje recibido -> Hola CODEHERO. Sun Mar 09 2014 01:05:31 GMT-0430 (VET)
Mensaje recibido -> Hola CODEHERO. Sun Mar 09 2014 01:05:34 GMT-0430 (VET)
Mensaje recibido -> Hola CODEHERO. Sun Mar 09 2014 01:05:35 GMT-0430 (VET)
```

Integridad

Puedes probar detener el **consumidor** y seguir enviando mensajes. En efecto el mensaje ha sido enviado a la cola y este no se ha perdido, en cuanto vuelvas a iniciar tu **consumidor** verás los mensajes encolados fluir.
Antes de levantar nuevamente el **consumidor** puedes probar entrar a la página administrativa de RabbitMQ para observar aquellos mensajes que se encuentran esperando y el flujo de entrada de ellos al servidor:



Además de poder echarles un vistazo desde esta misma interfaz ubicada en la parte inferior:



Escalabilidad

En caso que tengas un flujo de mensajes muy alto tan solo debes agregar más poder, levanta más consumidores y RabbitMQ se encargará de balancear la carga entre ellos siempre manteniendo un orden lógico en los mensajes y evitando que estos se pierdan o sean procesados más de una vez.
Para probar esto en la práctica, levanta varios **consumidores** y en lugar de utilizar el **publicador** que hemos probado hasta ahora, prueba el **publicador agresivo**. De esta manera podrás observar lo fácil que es escalar tu solución para soportar altos volumenes de peticiones.

Conclusión

Esta semana pudimos arañar la superficie de posibilidades con RabbitMQ, observamos su naturaleza asíncrona y algunas de las ventajas que su funcionamiento nos ofrece.

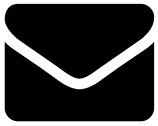
El enfoque productivo más común es aquel que se basa en suscripciones a contenidos mejor conocido como el modelo Pub/Sub, este por medio de un intercambiador distribuirá el mensaje para ser consumido por múltiples consumidores, algo como los *feeds*.
Otro enfoque importante es aquel basado en un procedimiento que se compone de varias tareas pesadas las cuales suelen ser síncronas lo cual obliga al usuario a esperar a que una tarea termina para proseguir, aplicaciones comunes son la gestión de imagenes y programación de tareas de corrida prolongada.

Por Jonathan Wiesel

Conoce más sobre este autor aquí (/author/jonathan.html)

 (https://github.com/codeheroco)  (https://twitter.com/codeheroblog) 

(https://facebook.com/codeheroblog)  (https://plus.google.com/u/0/+CodeheroCo) 

(https://linkedin.com/groups/Codehero-5108125)  (http://codehero.co/feed.xml) 

(mailto:heroes@codehero.co)  (https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=6TXLZX8PN8978)
