Queues With RabbitMQ

Lorna Mitchell, IBM

Introducing Queues

Use queues to:

- asynchronously process tasks in your (existing) application
  - e.g. sending email, processing uploads
- provide loose coupling points
  - e.g. delegate heavy tasks, split ownership/technology
- enable parts of the system to scale appropriately
  - e.g. an event in one system causes a thousand other actions

Queue Tools

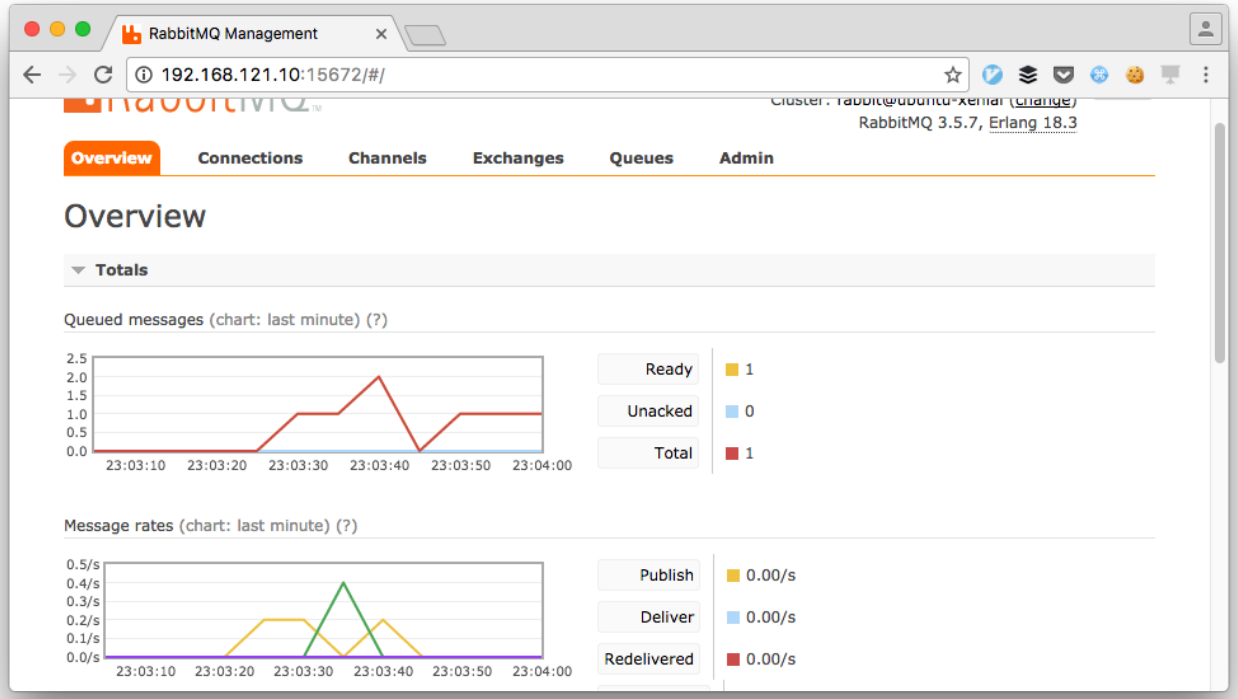
A selection of queue-ish tools

- RabbitMQ* http://www.rabbitmq.com/
- Gearman http://gearman.org/
- Beanstalkd http://kr.github.io/beanstalkd/
- Kafka https://kafka.apache.org/
- Redis https://redis.io/

* RabbitMQ is used in today's examples

Getting To Know RabbitMQ

Management Plugin



https://www.rabbitmq.com/management.html

Getting To Know RabbitMQ

Some vocabulary

- **broker**: the RabbitMQ instance
- **exchange**: where to send the messages to
- **queue**: where messages wait to be processed
- **binding key**: rules for which messages go into this queue
- **message**: the data to be processed
- **routing key**: message route information
- **consumer**: worker script to processes the messages

RabbitMQ Exchanges

Exchanges are the routing logic of RabbitMQ

Messages go to exchanges and the exchanges put them into the correct queues for storage

Types Of Exchange

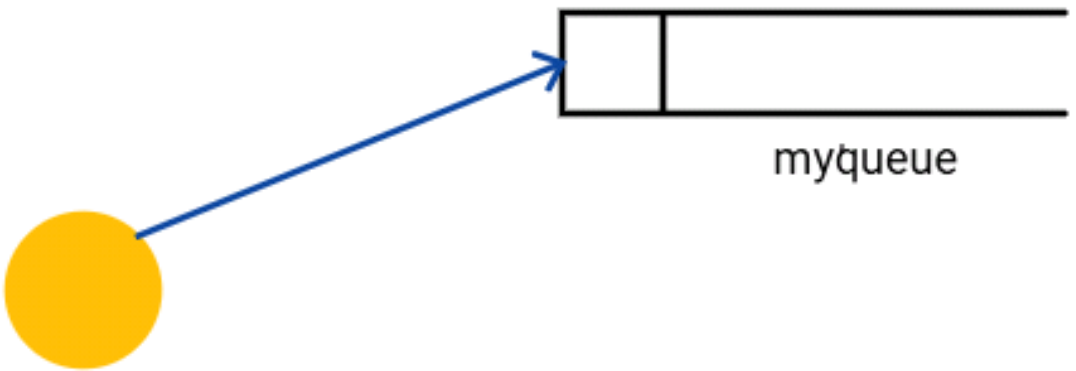
Direct: a given routing key puts messages onto the matching queue(s)

Topic: queues are bound by key, and messages are routed to as many queues as their routing key matches

Fanout: messages go to all queues bound to this exchange

Default Exchange

There is a default exchange in RabbitMQ



Its name is '' and routing is on queue name

RabbitMQ Queues

Queues can:

- have wildcards in binding keys
- be durable (messages have their own durability)
- have a maximum length
- be configured with a "dead letter exchange"

RabbitMQ Messages

Messages have:

- a body consisting of a string of data (JSON is common)
- additional data, including TTL (Time To Live) and message durability
- may have priority information

RabbitMQ and PHP

Sample code here:
https://github.com/lornajane/queues-with-rabbitmq

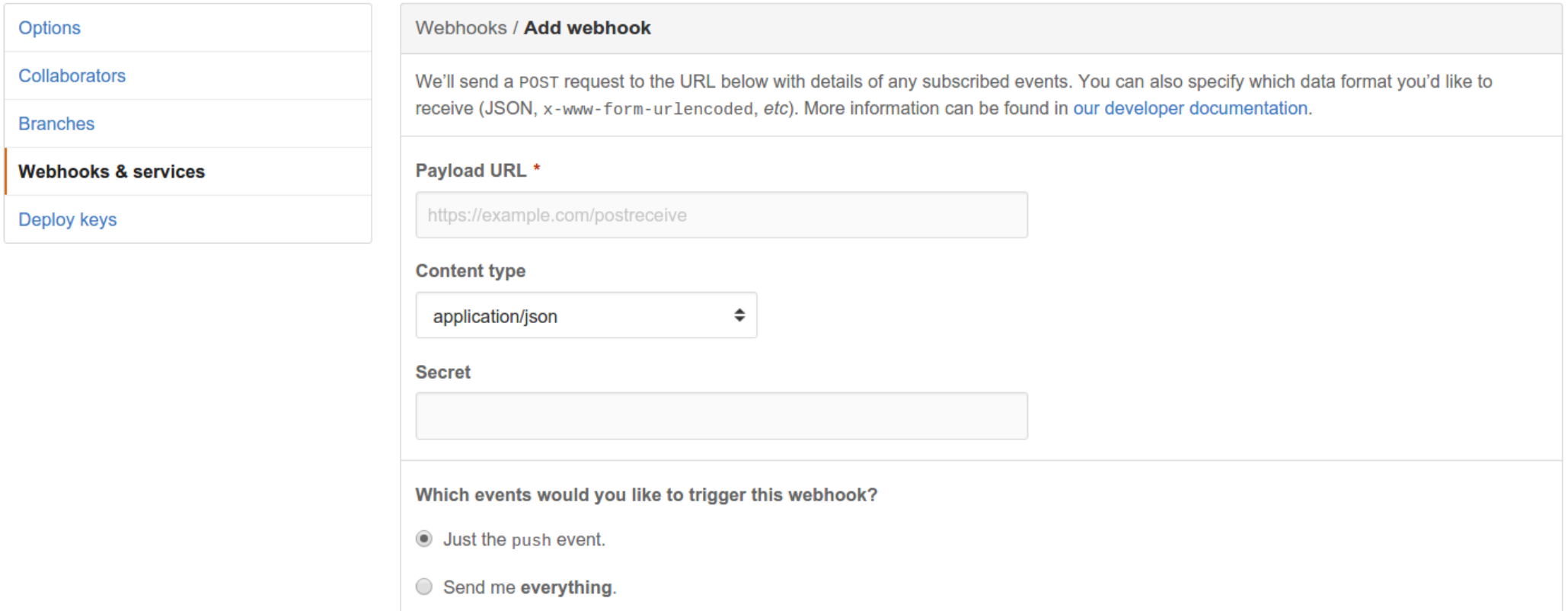
RabbitMQ and PHP

Best library: https://github.com/php-amqplib/php-amqplib

```
composer require php-amqplib/php-amqplib
```

- dependencies include `bcmath` and `mbstring`

Webhooks on GitHub



Ngrok for Local Webhooks

https://ngrok.com/ - secure tunnel to your dev platform

Use this tool to:

- webhook into code running locally
- inspect the request and response of the webhook
- replay requests and see the responses

PHP Producer

Receive webhooks from GitHub, add to queue

```php
require "vendor/autoload.php";
$input = file_get_contents("php://input");
$data = json_decode($input, true);

$rabbit = new PhpAmqpLib\Connection\AMQPStreamConnection(
    'localhost', 5672, 'guest', 'guest');
$channel = $rabbit->channel();

$channel->queue_declare('pushes', false, true, false, false);
$message = new PhpAmqpLib\Message\AMQPMessage(
    $input, ["delivery_mode" => 2]);
$channel->basic_publish($message, '', 'pushes');
```

PHP Consumer

Process a queue with PHP

```
1 require "vendor/autoload.php";
2 $rabbit = new PhpAmqpLib\Connection\AMQPStreamConnection(
3     'localhost', 5672, 'guest', 'guest');
4 $channel = $rabbit->channel();
5
6 $channel->queue_declare('pushes', false, true, false, false);
```

PHP Consumer

Process a queue with PHP *(continued)*

```
 1 $process = function ($message) {
 2     $data = json_decode($message->getBody(), true);
 3     // do message processing here
 4     $message->delivery_info['channel']->
 5         basic_ack($message->delivery_info['delivery_tag']);
 6 };
 7
 8 $channel->basic_consume('pushes', '', false, false,
 9     false, false, $process);
10
11 while(count($channel->callbacks)) { $channel->wait(); }
```

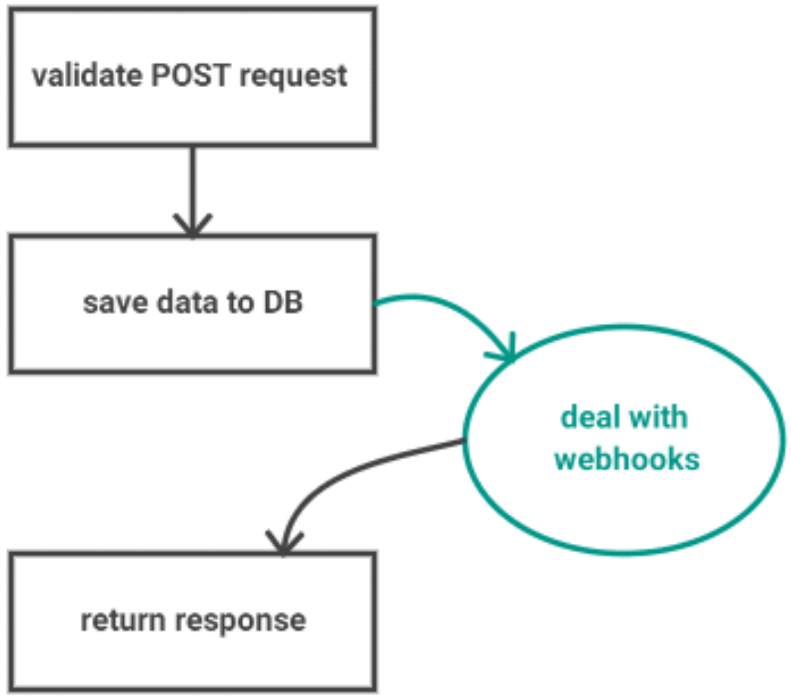
Example App: Webhooks

Example App: Webhooks

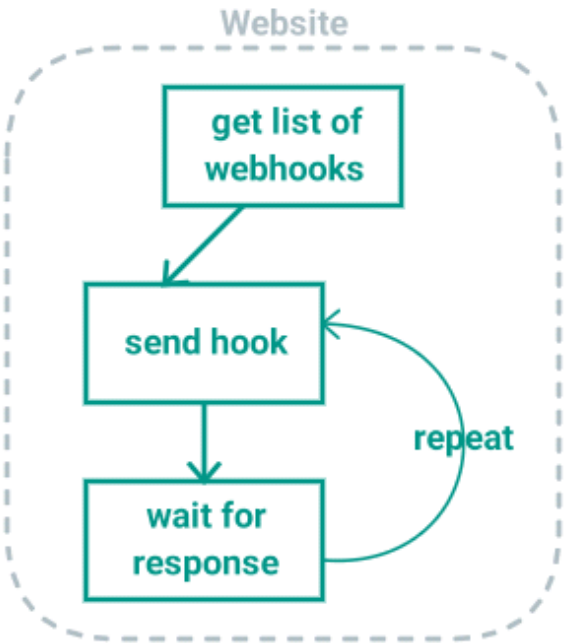
(*code:* https://github.com/ibm-watson-data-lab/guestbook)

A simple guestbook application is extended to allow webhook notifications of new comments.

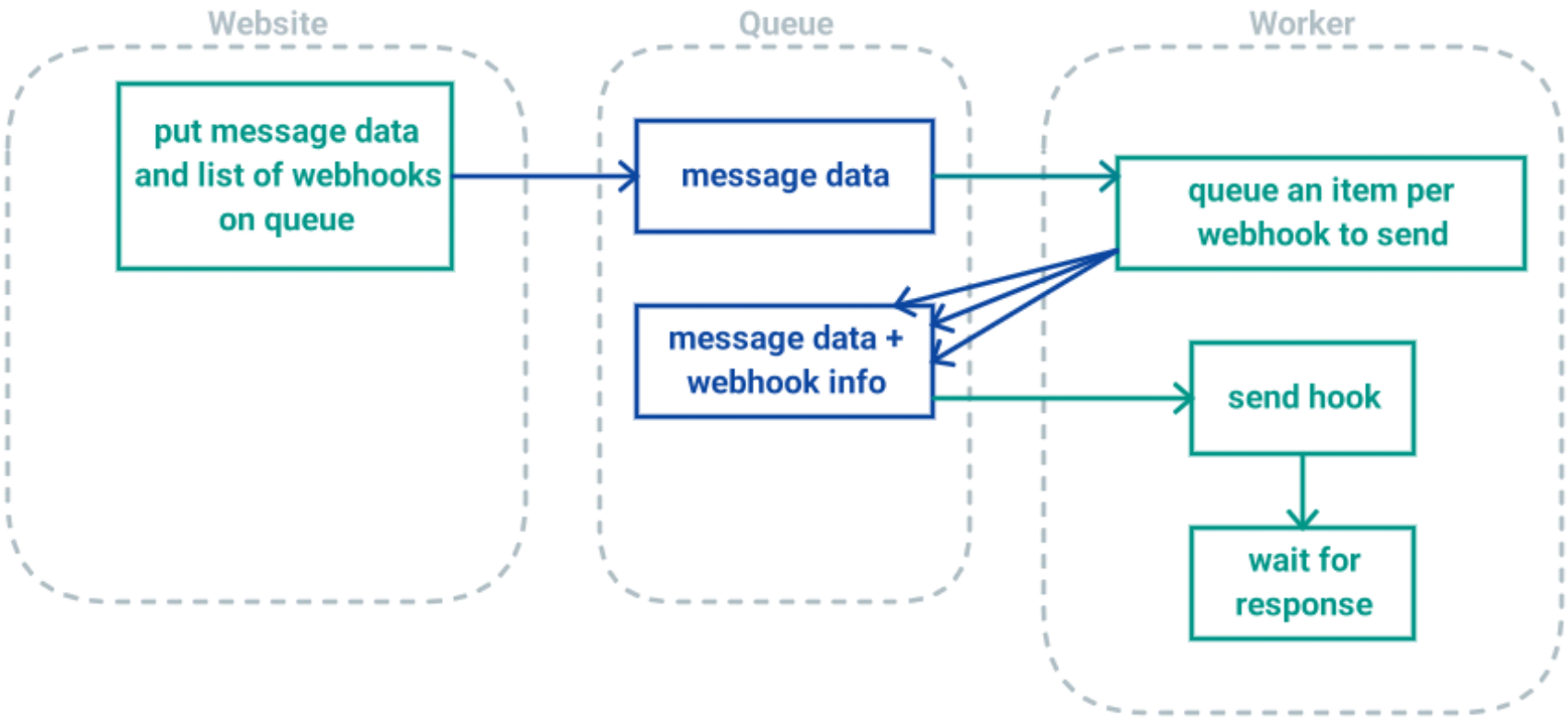
Example App: Webhooks



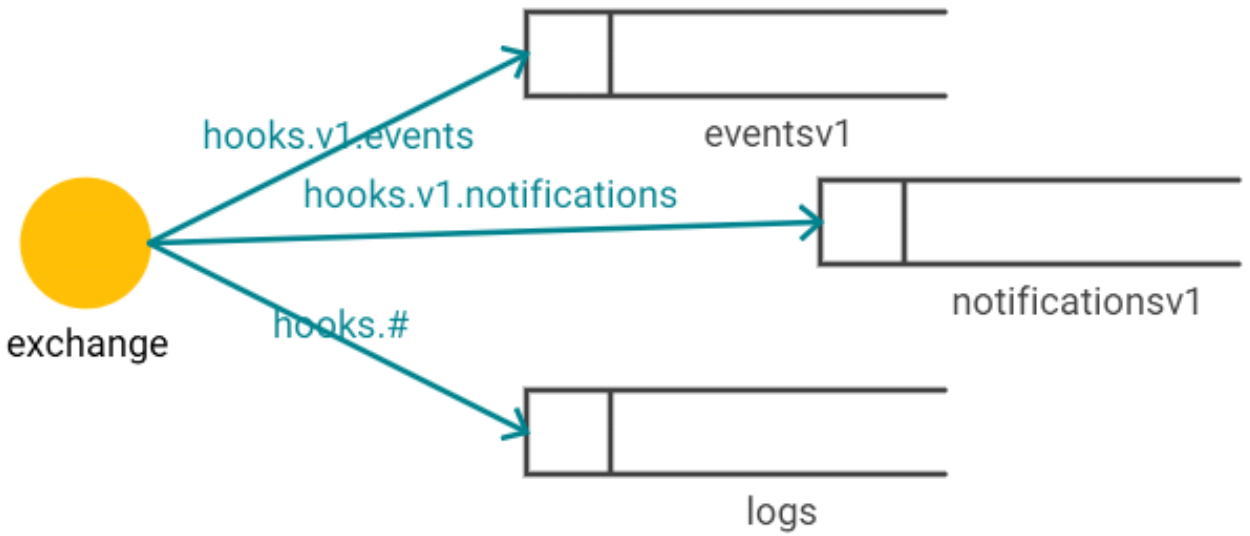
Example App: Webhooks



Example App: Webhooks



Example App: Webhooks



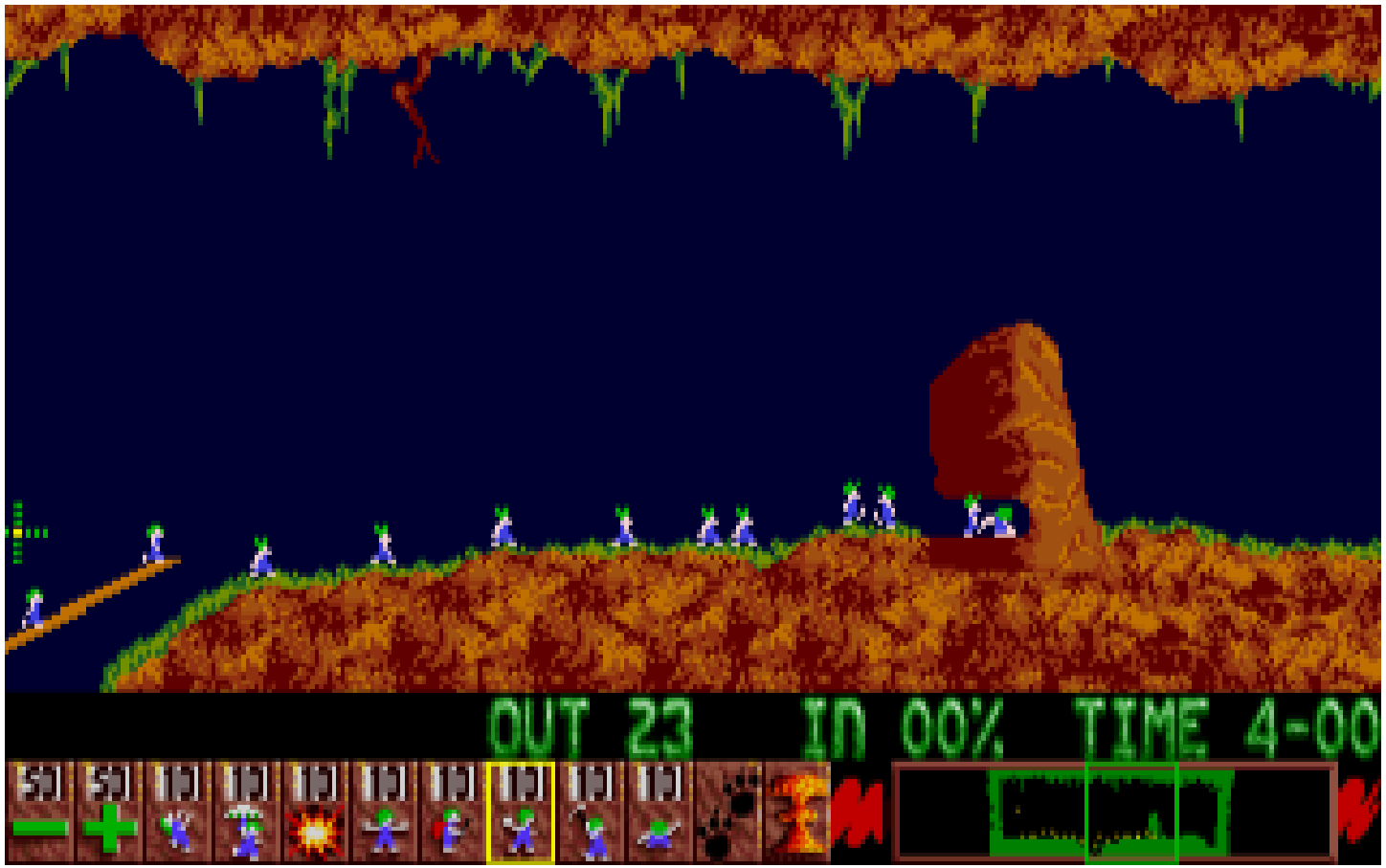
Processing Messages

Creating Workers

Workers are disposable!

- if things go wrong, exit
- separate tool to monitor/restart as needed
- beware long-running process hazards
- everything processed "at least once" (but maybe more than once, and in any order...)

Independent Workers



Independent Workers

For best results:

- awesome, aggregated logging
- monitoring: queue size, worker uptime, processing time
- miniminum viable dependencies

Completed Messages

- acknowledge when messages are processed successfully
- can acknowledge failure
    - reject the message
    - optionally: requeue

Dead Letter Exchanges

- reject without requeue
- exceed TTL
- queue length exceeded

Retries

Implement your own logic to handle retries

Create a *new* message with:

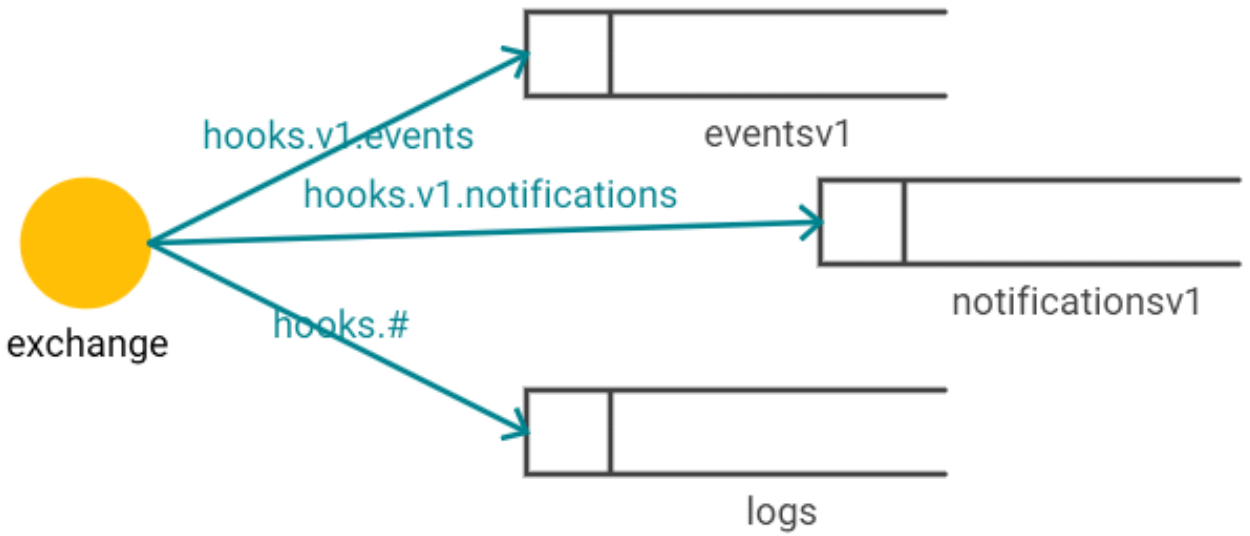
- all the existing message contents
- plus some metadata such as retry count or backoff time

Feedback Mechanisms

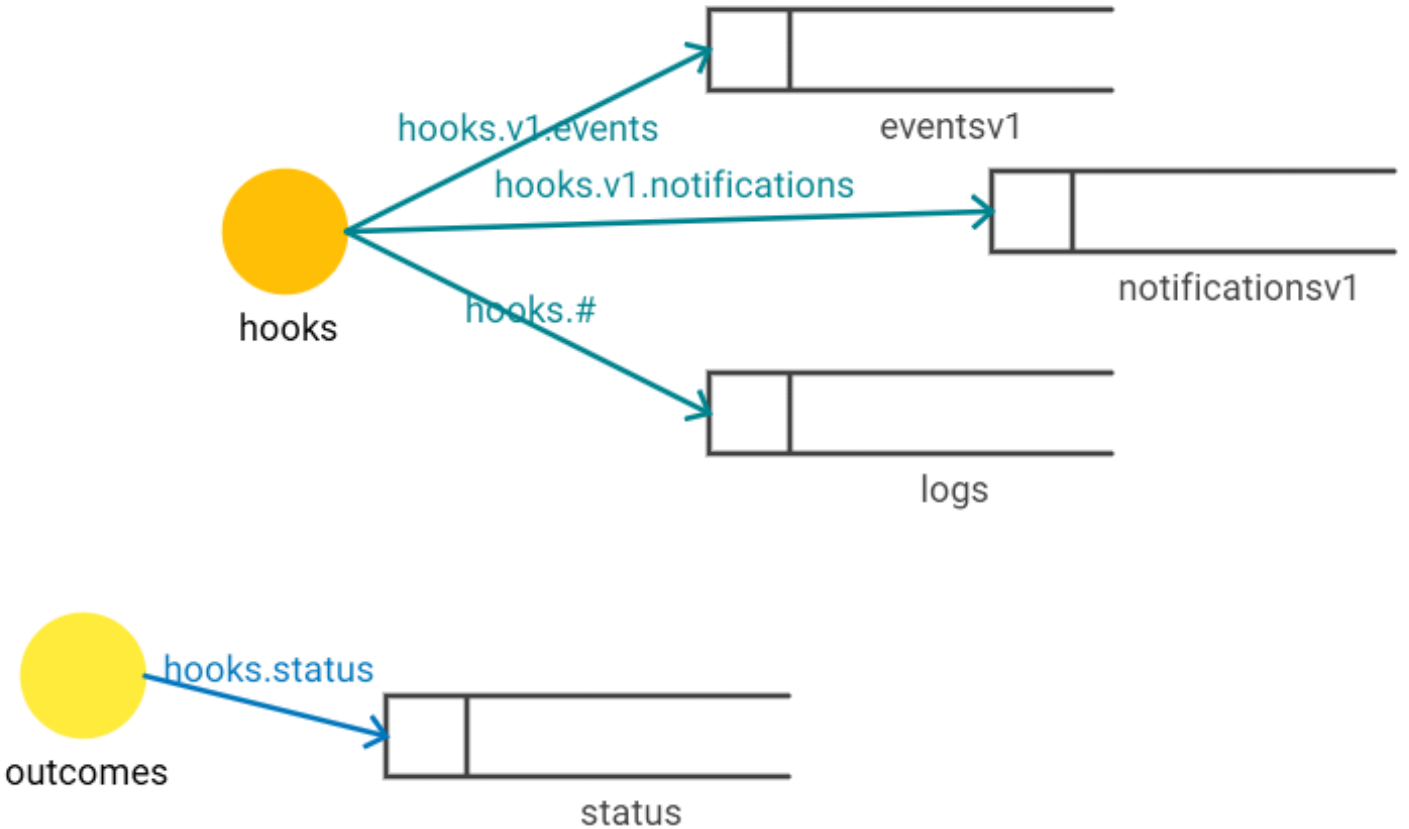
Rabbit is fire-and-forget; work is delegated

Common pattern: return queue to put updates into for the original producer then to consume.

Example App: Webhooks



Example App: Webhooks



Queues With RabbitMQ

Queues With RabbitMQ

Queues are awesome for scalability and robustness

RabbitMQ is open source, lightweight and fast

Queues help us meet the requirements for modern applications

Thanks!

Resources:

- RabbitMQ: https://www.rabbitmq.com/
- Try it: https://ibm.com/cloud
- Blog: https://lornajane.net
- Code: https://github.com/ibm-watson-data-lab/guestbook

- `@lornajane`
- lorna.mitchell@uk.ibm.com