

# Whitepaper: Using AMQP based messaging to manage a cloud.

Presented by:

# STORMMQ

## Table of Contents:

1.0 Introduction .....	2
2.0 The Problem faced by StormMQ .....	3
2.1 Clouds are Complex .....	2
2.2 Clouds Generate lots of Important Information .....	3
3.0 StormMQ's Ideal Solution .....	4
3.1 Managing Information from the Cloud .....	4
3.2 Active Firewalls .....	4
3.3 Live Logging .....	4
3.4 Live Statistics .....	4
3.5 Statistics Mining .....	4
3.6 SMS & EMail Alerts .....	5
4.0 What StormMQ Did .....	5
4.1 Message Queuing is Ideal .....	5
4.2 WebSockets empower Live Monitoring .....	5
4.3 Integrating Configuration Management .....	5
4.4 Feedback: Actuators .....	5
4.5 Monitoring without a Server Database .....	6
5.0 In conclusion .....	6

## 1.0 Introduction

This whitepaper is designed to provide a pragmatic guide as to the challenges that were faced by us ([StormMQ](#)) when we started out on our vision to provide a cloud based service that hitherto was always designed, built and supported by in house by IT departments usually using expensive proprietary software.

The service is a very specific infrastructure piece of middleware allowing disparate computers to communicate with each other through an emerging standard known as [Advanced Message Queuing Protocol \(AMQP\)](#).

This machine to machine communication, or messaging, is usually referred to as “Message Queuing” and has been used for a long time - especially in large investment banks and telco companies. However because of the high cost of proprietary licenses and huge ongoing fees for support and maintenance, this kind of middleware has had limited acceptance beyond such companies. AMQP is set to change this by creating an open standard that any company can write their own version of software or choose between different variants knowing that they will be able to communicate with each other, in the same way that emails use SMTP or websites use HTTP. Having choice lowers cost and indeed there are open source offerings of AMQP. However open source still represents high costs in terms of consultancy, implementation support and maintenance.

StormMQ has taken things further to create a Managed Service for Message Queuing or “Cloud” based Message Queuing. In so doing we had various challenges and problems to overcome that we are happy to share in this White paper.

## 2.0 The Problem faced by StormMQ

### 2.1 Clouds are Complex

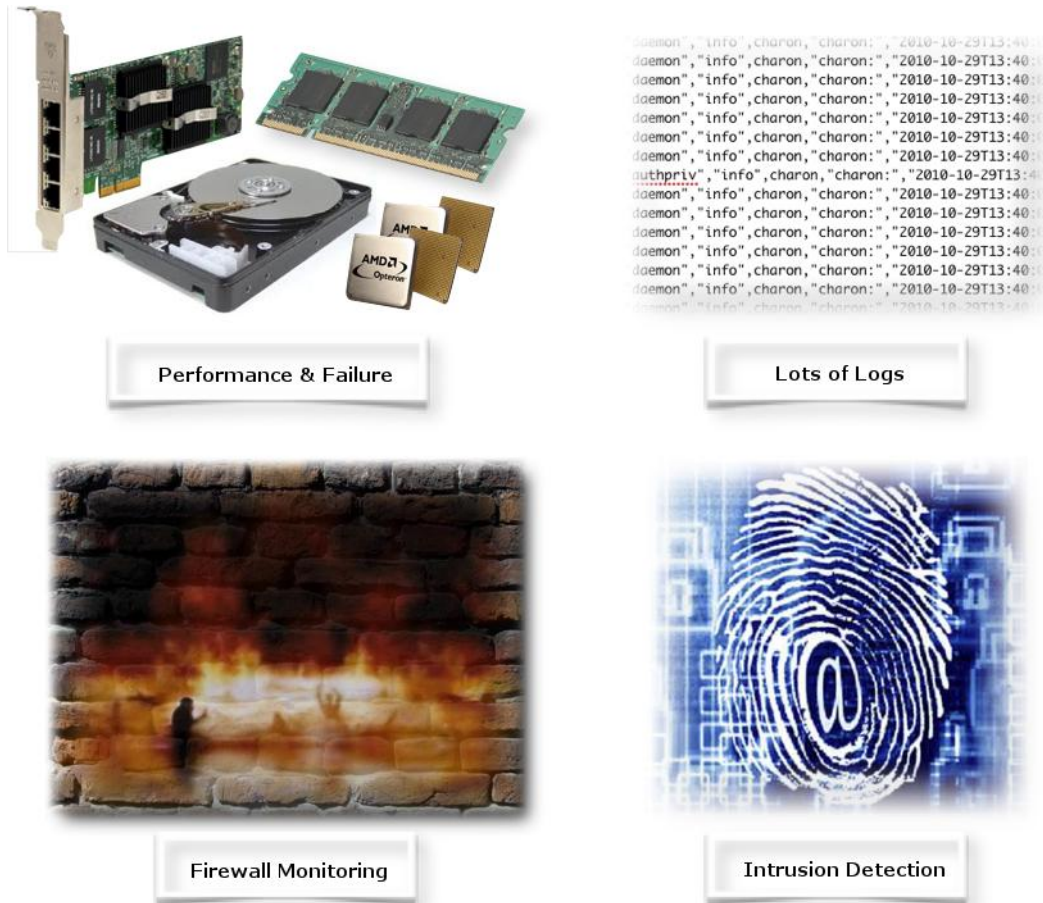
Running a cloud requires a lot of hardware, software and man hours: servers, operating systems and administrators. Running a cloud profitably requires that those resources are used to maximum effect; so that not a CPU cycle is wasted, not one piece of hardware sits in storage too long and any one administrator can manage as much hardware as possible. In essence, a cloud needs to run itself with minimal intervention. Adding hardware, changing software and preventative maintenance must be automated, self-healing and self-updating: the nirvana of zero configuration!



Early on, StormMQ faced this problem. Our cloud is quite complex under the covers. We use double-ring firewalls, with fire walling of every server and groups of equipment. Every network connection uses IPSec security. And the disk arrays, well, they're designed for redundant failure. In our early days, we had to spend our money wisely - but also avoid any significant outage, as it could lose us the business we had built up.

## 2.2 Clouds Generate lots of Important Information

Of course, such a set up generates a lot of information from a diverse range of places:-



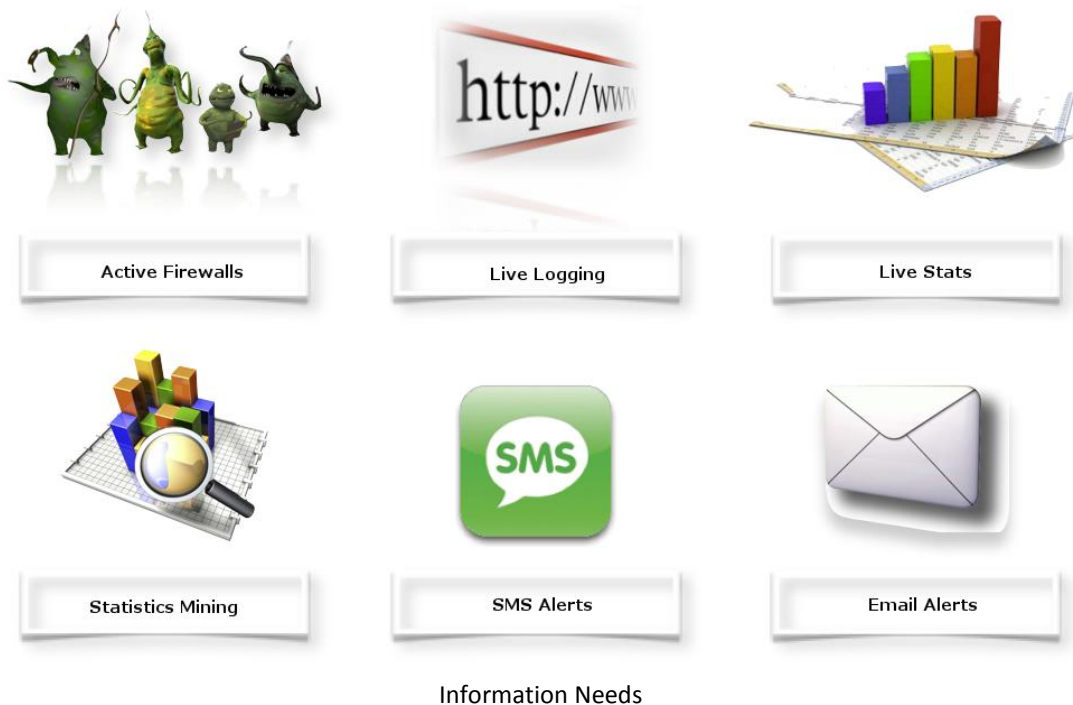
Sources of Information in a Cloud

For instance, several of StormMQ's servers have over 2,500 firewall incidents an hour. We needed active firewalls that could adapt, and approaches to sort the wheat of true problems from the chaff of background noise. Initially, we used the standard linux tools, such as syslog. We stored everything in a large central log database, and then tried to use standard GUIs and analysis tools. It didn't work well. The volumes were too great, and the tools too cumbersome: we failed in our need to minimise administration.

### 3.0 StormMQ's Ideal Solution

#### 3.1 *Managing Information from the Cloud*

The information we were going to be generating needed to be sorted and used to provide six key things:-



#### 3.2 *Active Firewalls*

We need to continuously sort useful information, and adapt firewall rules to match to keep the bad guys out; essentially, a feedback approach. However, we couldn't do that until we knew what was going on.

#### 3.3 *Live Logging*

We need to be able to fire up a web browser, from anywhere we were, and actively see what was going on. To effectively develop our solution would require many iterations - one of which was always going to be letting a human to make decisions.

#### 3.4 *Live Statistics*

It's great to be able to do preventative maintenance, to know when a disk is 90% full or a CPU is saturated, and see if it's a one off or occurring regularly, or in concert with something else. So being able to produce live charts of data in the browser mattered - and being able to do in such a way that the source and structure of the data didn't have to be known in advance.

#### 3.5 *Statistics Mining*

Of course, not all trends can be observed over a few minutes. The value of being able to see the change in the use of a disk allows the development of rules tuned to what actually happens in our cloud. We'd need a way of mining data, either for human analysis or for a stateful 'program' to send alerts.

### **3.6 SMS & Email Alerts**

We wanted alerts, of course - but not death by text, which wouldn't help. Yet no alerting system can know what is important in advance. To avoid the potential chicken and egg that might mean we needed a simple alerts system that could suppress messages when the volume was too much.

## **4.0 What StormMQ Did**

We realised that we already had the technology to make these needs happen. By using the full features of StormMQ's AMQP message queuing, [Kaazing's WebSockets platform](#), and a secure continuous deployment platform, we devised an elegant, simple solution.

### **4.1 Message Queuing is Ideal**

Message queuing is ideal for managing the flow of information. We chose a classic architecture of 'pipes and filters' - essentially lots of little nodes, each a tiny program, doing one thing. In this world, a node acts like a valve between a couple of pipes. Each 'unit of resource' capable of hosting a message queuing client (and AMQP fits in 32K) had a tiny 'monitoring' client node added. Wherever possible, these sat as close to the source of data as possible, and did as much processing as possible. This maximised the use of CPU cycles, and meant nodes could be very specific to the resource they monitored. They then sent out event messages. Using AMQP meant we could be agnostic to the Operating System and programming language; we had nodes in C, Python, Ruby, Bash and Java.

### **4.2 WebSockets empower Live Monitoring**

Other nodes could subscribe to event messages of interest, and handle them. Some of these might store the data (a statistics mining tool, for example). Others might only have a temporary interest - a web browser looking at live log events for a specific server. These used Kaazing WebSockets to make the information live. In effect, these nodes acted almost like a multimeter on an electrical connection - a probe to check something.

### **4.3 Integrating Configuration Management**

More sophisticated nodes use configuration management information to generate events themselves. For example, bandwidth is a major cost for us. Different customers have different allocations, and 'policies' for using it - some have a fixed quota, whilst others have rules around 'burst' if other customers aren't using theirs.

### **4.4 Feedback: Actuators**

Some nodes are 'actuators'. They respond to events to turn a tap off, or the like. For example, we do this for bandwidth management. The actuator is simple - if it had physical form it'd look like a faucet on a pipe. The event message turns its handle. It has no need to know the circumstances of why it's changing. Another place we use this is with firewalls, to provide active change.

Not all things can be done like this. Some rules can only be added after observation - the problem is a classic evolutionary one. And a potential problem of having tiny nodes embedded in the resource framework with specific rules is the cost of change. Traditionally, this is very high, as an administrator would have to log on to individual servers (and remember which nodes we where, too, so the risk of error is magnified). We used an elegant secure deployment platform, using meta-packages. This made nodes selfupdating. Changes to deployment where controlled using software 'keys' to restrict access.

Our entire configuration is managed using the concept of 'company, system, environment'. We designed it so that it can be managed as a policy, audited, stored in source control or used dynamically (egg queues created on use). Changes to configuration are broadcast, allowing sophisticated nodes to take this into account when generating events.

#### ***4.5 Monitoring without a Server Database***

It'd seem that adding servers would be hard in this world. Surely we'd need somewhere central to store server details? Not really. One server should, must, look like another. So when a server is added, it just starts sending its event messages out. Any monitoring tool can notice this, and start tracking it. If the event messages stop - then generate an alert event.

### **5.0 In Conclusion**

The essence of the services that we provide we also use to monitor our own Cloud based environment to ensure we conform to our Service Level Agreement commitments - while maximising available resources to ensure appropriate margins are maintained to create a profitable business. Message Queuing can be used in so many different ways and we hope that this White Paper gives a particular insight into just how the concept can be used for managing Cloud based services. What we have put in place, in terms of instrumentation and configuration management, allows us to deliver a totally robust, secure and scalable enterprise Message Queue solution as a Managed Service. This allows companies to simply subscribe to us for a monthly fee in order to have very affordable machine to machine messaging that adheres to an open standard of AMQP, thus avoiding any vendor lock in that up to now has been the norm for this type of middleware solution.