Iron.io

Whitepaper

Top 10 Uses of a Message Queue

Asynchronicity, Work Dispatch, Load Buffering, Database Offloading, and More

From Monolithic Apps to Distributed Architectures

Language and application frameworks like Ruby on Rails, Python and Django, and PHP and Zend have transformed the use of cloud infrastructure and drastically increased cloud adoption for common types of applications. But apps are becoming more complex. There are more interfaces, greater expectations on response times, increasing connections to other systems, and lots more processing around each event.

This next shift in cloud development is less about building monolithic apps and more about creating highly scalable and adaptive systems. As cloud and hybrid applications evolve, more capable architectures are called for. But these architectures don't need to be refactored in, nor are they incompatible with application frameworks.

More and more developers are starting out with a distributed system in mind and using message queues and asynchronous processing at the onset to gain scale and make applications easier to build and manage. Besides the ability to scale workloads more easily, one of the bigger advantages of starting with a distributed app in mind is it makes it easier to build features and expand capabilities.

Instead of having one process directly invoke another process, it's much cleaner to use a message queue to keep the processes separate and independent. Need to post a notification about an event – easy, just add a message to a queue and a separate notification process can pick up messages from the queue and take care of the task.

By putting in a middle layer between processes – instead of invoking the second process directly – the processes become independent of each other. The first process above doesn't need to know how to post a notification and it doesn't have to monitor the process flow of that second task. It's just puts a message on the queue and continues on with its processing.

The other processes can also just do their work independently, creating a system that's easy to grow and maintain. Message queues also provide an easier and more rational way of handling exceptions. If the notification process fails, the corresponding message to the action can be put back on the queue for retrying or placed in another queue specifically designed for handling exceptions and escalating issues. (Try doing exception handling when processes are tightly coupled.)

Putting in this type of design up front results in code that's far less brittle and much easier to maintain and expand. Which is a huge benefit regardless of whether you're just getting to market or are already a hit and growing 25% month-to-month.

"You can find a message queue in nearly every major architecture profile on High Scalability.

Historically they may have been introduced after a first generation architecture needed to scale up from their two tier system into something a little more capable (asynchronicity, work dispatch, load buffering, database offloading, etc).

If there's anything like a standard structural component, like an arch or beam in architecture for software, it's the message queue."

- High Scalability

Try IronMQ Today

Iron.io provides an easy to use highly available elastic message queue. Try it for free today:

www.iron.io

For additional information, contact sales at:

1-888-939-4623
Or by email at:
sales@iron.io

Types of Message Queues

Message queues are the bonds that tie distributed architectures together. They're used to transmit data between the component parts of the architecture. The general concept is that chunks of data (messages) are put into a group (a queue) and then removed from that group in a predefined order, most often on FIFO (first in first out) basis but also sometimes on a FILO (first in last out) basis.

Pull Queues and Push Queues

Message queues generally come in two types – pull queues and push queues.

Pull queues are queues that ask clients to periodically check for messages on a queue. If a message is found, the queue will give it to the client and remove it from the queue. If a message is not found, the client just tries again after a set period of time. Pull queues are great for data flows that are heavy or consistent because there's a high chance that the request will yield a message, not a waste of bandwidth and processing power, but the queue won't bring down the client by delivering more messages than the client can handle.

Push queues are queues that inform subscribers when a message is added to the queue. This means that data is processed more promptly because subscribers know about it immediately, but it also means that the queue and client have to do more work, sending and receiving the data. Push queues are great for data flows that are sporadic, because the data comes infrequently. Knowing about the data as soon as it comes is worth the tradeoff of the minimal increase in extra processing.

Long-polling is an addition to a pull queue in that a request will yield one or messages upon a get request. Long-polling adds an additional capability in that the message request is keep the open for a certain period of time. If a message arrives in the queue during that period of time, the request will return with the message.

Benefits of Message Queuing Services

Adding message queues makes sense for almost any production-scale application, but it makes even more sense if there are net gains in terms of installation and operation. Adding a messaging layer by having to stand up servers and deal with reliability (servers, API connections, monitoring, log files, etc.) is not an easy task for most teams, even ones with dedicated sysops team.

When message queues are easy to set up, simple to use, highly available, and extremely reliable, a whole new world opens up. The analogy is the generation of power. The progression moved from water-driven mill stones to individual coal-fired factories and ultimately to industrial-scale power plants and transmission lines. This last step – the industrialization of power – transformed the industry and the world. It lowered the cost of building and making things, revolutionized cities, factories, and homes, and ushered in new inventions, services, and businesses.

Similarly, by plugging into elastic messaging services, developers no longer have to maintain large sets of queues running on multiple servers. In an elastic world, service providers like Iron.io take on the responsibility for seamlessly managing servers, API endpoints, multiple zones, and other infrastructure resources and abstracting away most physical constraints.

Benefits of the shift to cloud-based messaging include:

- Speed to market: applications and systems can be built much more quickly
- Reduced complexity: reduced risk/overhead in critical but non-strategic areas
- Increased scalability: ability to seamlessly scale throughput and functionality

Top 10 Uses of a Message Queue

1. Decoupling

It's extremely difficult to predict at the start of a project what the future needs of the project will be. By introducing a layer in between processes, message queues create an implicit, data-based interface that both processes implement. This allows you to extend and modify these processes independently, by simply ensuring they adhere to the same interface requirements.

2. Redundancy

Sometimes processes fail when processing data. Unless that data is persisted, it's lost forever. Queues mitigate this by persisting data until it has been fully processed. The put-get-delete paradigm, which many message queues use, requires a process to explicitly indicate that it has finished processing a message before the message is removed from the queue, ensuring your data is kept safe until you're done with it.

3. Scalability

Because message queues decouple your processes, it's easy to scale up the rate with which messages are added to the queue or processed; simply add another process. No code needs to be changed, no configurations need to be tweaked. Scaling is as simple as adding more power.

4. Elasticity & Spikability

When your application hits the front page of Hacker News, you're going to see unusual levels of traffic. Your application needs to be able to keep functioning with this increased load, but the traffic is anomaly, not the standard; it's wasteful to have enough resources on standby to handle these spikes. Message queues will allow beleaguered components to struggle through the increased load, instead of getting overloaded with requests and failing completely.

5. Resiliency

When part of your architecture fails, it doesn't need to take the entire system down with it. Message queues decouple processes, so if a process that is processing messages from the queue fails, messages can still be added to the queue to be processed when the system recovers. This ability to accept requests that will be retried or processed at a later date is often the difference between an inconvenienced customer and a frustrated customer.

Top 10 Uses of a Message Queue

6. Delivery Guarantees

The redundancy provided by message queues guarantees that a message will be processed eventually, so long as a process is reading the queue. On top of that, IronMQ provides an only-delivered-once guarantee. No matter how many processes are pulling data from the queue, each message will only be processed a single time. This is made possible because retrieving a message "reserves" that message, temporarily removing it from the queue. Unless the client specifically states that it's finished with that message, the message will be placed back on the queue to be processed after a configurable amount of time.

7. Ordering Guarantees

In a lot of situations, the order with which data is processed is important. Message queues are inherently ordered, and capable of providing guarantees that data will be processed in a specific order. IronMQ guarantees that messages will be processed using FIFO (first in, first out), so the order in which messages are placed on a queue is the order in which they'll be retrieved from it.

8. Buffering

In any non-trivial system, there are going to be components that require different processing times. For example, it takes less time to upload an image than it does to apply a filter to it. Message queues help these tasks operate at peak efficiency by offering a buffer layer--the process writing to the queue can write as fast as it's able to, instead of being constrained by the readiness of the process reading from the queue. This buffer helps control and optimize the speed with which data flows through your system.

9. Understanding Data Flow

In a distributed system, getting an overall sense of how long user actions take to complete and why is a huge problem. Message queues, through the rate with which they are processed, help to easily identify under-performing processes or areas where the data flow is not optimal.

10. Asynchronous Communication

A lot of times, you don't want to or need to process a message immediately. Message queues enable asynchronous processing, which allows you to put a message on the queue without processing it immediately. Queue up as many messages as you like, then process them at your leisure.

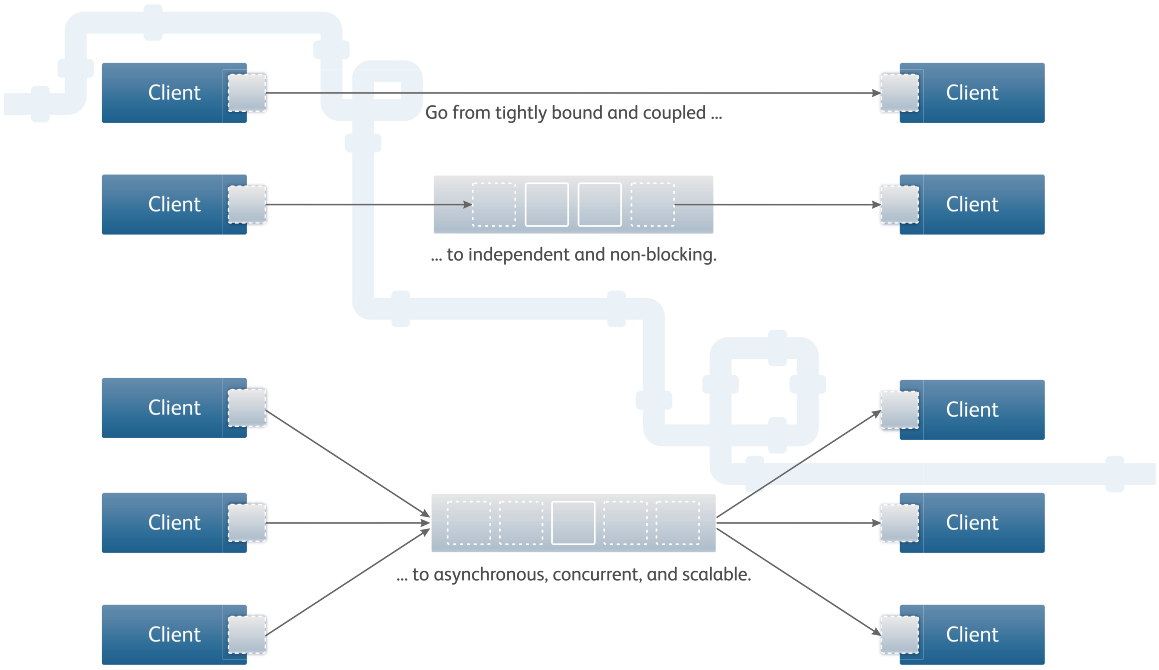
Iron.io

IronMQ

High Availability Message Queuing

Iron.io provides a highly available message queue called IronMQ. It features push queues, error queues, alerts, retries, and more. IronMQ can be used to power asynchronous messaging, work dispatch, load buffering, and database offloading and can serve as the foundation for cloud-based service-oriented architectures.

IronMQ is available as a scalable cloud-based service that operates in multiple zones and clouds for reliability and high availability. The use of cloud services creates efficiencies and agility because it eliminates having to stand up and maintain infrastructure components, not to mention making them redundant and failsafe.
(IronMQ can also be installed in private clouds and even on-premise, providing enterprise-grade and carrier-grade capabilities. For more information on these options, contact our sales and support group.)



Visit www.iron.io today to start queuing instantly

Iron.io

Iron.io is the maker of IronMQ, an industrial-strength message queue, and IronWorker, a scalable task processing/worker platform. Iron.io's products are designed for building distributed cloud applications quickly and operating at scale. They are easy to use, always on, and highly available.