**Azure Storage queues and Service Bus queues**

**07/17/2017**

**Issue:** Health Fitness is experiencing periods of overload from massive client submissions and requests for updates, inserts, and reporting protocols. One of our major clients is the State of Nebraska and their submissions, combined with a dozen other large clients, overwhelm the server bank 2 to 3 times weekly.

**Offer: Implementation of**

* **Pre-calculated cubes** eliminates the costs of running the very expensive reporting stored procedure(s). Eliminates SP calculations.
* **Service Bus Queues** Processes thousands of requests in a controlled FIFO process and can be customized

**Assignment:**

* **Dale Miller**
* **Scott Montgomery**

**Follow-up 12/15/2017**

In the last 90 days, we have not had one failure due to batch submission overload or request for reports and/or data. The average run time required to generate a client usage and statistics report decreased from 18 minutes per to under 1 minute. The proposed corrective actions seem to have greatly reduced failure and server load.

The below is provided by Microsoft and is a workable solution for the problems we are experiencing with server overload and long waits/timeouts of client requests for reporting and data.

**Storage queues**, which are part of the [Azure storage](https://azure.microsoft.com/services/storage/) infrastructure, feature a simple REST-based GET/PUT/PEEK interface, providing reliable, persistent messaging within and between services.

**Service Bus queues** are part of a broader [Azure messaging](https://azure.microsoft.com/services/service-bus/) infrastructure that supports queuing as well as publish/subscribe, and more advanced integration patterns. For more information about Service Bus queues/topics/subscriptions, see the [overview of Service Bus](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview).

While both queuing technologies exist concurrently, Storage queues were introduced first, as a dedicated queue storage mechanism built on top of Azure Storage services. Service Bus queues are built on top of the broader messaging infrastructure designed to integrate applications or application components that may span multiple communication protocols, data contracts, trust domains, and/or network environments.

**Technology selection considerations**

Both Storage queues and Service Bus queues are implementations of the message queuing service currently offered by Microsoft Azure. Each has a slightly different feature set, which means we can choose one or the other, or use both, depending on the needs of your particular solution or business/technical problem we are solving.

When determining which queuing technology fits the purpose for a given solution, solution architects and developers should consider these recommendations. For more details, see the next section.

As a solution architect/developer, **we should consider using Storage queues** when:

* Your application must store over 80 GB of messages in a queue.
* Your application wants to track progress for processing a message inside of the queue. This is useful if the worker processing a message crashes. A subsequent worker can then use that information to continue from where the prior worker left off.
* We require server side logs of all of the transactions executed against your queues.

As a solution architect/developer, **we should consider using Service Bus queues** when:

* Your solution must be able to receive messages without having to poll the queue. With Service Bus, this can be achieved through the use of the long-polling receive operation using the TCP-based protocols that Service Bus supports.
* Your solution requires the queue to provide a guaranteed first-in-first-out (FIFO) ordered delivery.
* Your solution must be able to support automatic duplicate detection.
* You want your application to process messages as parallel long-running streams (messages are associated with a stream using the [SessionId](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.sessionid) property on the message). In this model, each node in the consuming application competes for streams, as opposed to messages. When a stream is given to a consuming node, the node can examine the state of the application stream state using transactions.
* Your solution requires transactional behavior and atomicity when sending or receiving multiple messages from a queue.
* Your application handles messages that can exceed 64 KB but will not likely approach the 256 KB limit.
* You deal with a requirement to provide a role-based access model to the queues, and different rights/permissions for senders and receivers.
* Your queue size will not grow larger than 80 GB.
* You want to use the AMQP 1.0 standards-based messaging protocol. For more information about AMQP, see [Service Bus AMQP Overview](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview).
* You can envision an eventual migration from queue-based point-to-point communication to a message exchange pattern that enables seamless integration of additional receivers (subscribers), each of which receives independent copies of either some or all messages sent to the queue. The latter refers to the publish/subscribe capability natively provided by Service Bus.
* Your messaging solution must be able to support the "At-Most-Once" delivery guarantee without the need for you to build the additional infrastructure components.
* You would like to be able to publish and consume batches of messages.

**Comparing Storage queues and Service Bus queues**

The tables in the following sections provide a logical grouping of queue features and let you compare, at a glance, the capabilities available in both Azure Storage queues and Service Bus queues.

**Foundational capabilities**

This section compares some of the fundamental queuing capabilities provided by Storage queues and Service Bus queues.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Ordering guarantee | **No**   For more information, see the first note in the “Additional Information” section. | **Yes - First-In-First-Out (FIFO)**  (through the use of messaging sessions) |
| Delivery guarantee | **At-Least-Once** | **At-Least-Once**  **At-Most-Once** |
| Atomic operation support | **No** | **Yes** |
| Receive behavior | **Non-blocking**  (completes immediately if no new message is found) | **Blocking with/without timeout**  (offers long polling, or the ["Comet technique"](https://go.microsoft.com/fwlink/?LinkId=613759))  **Non-blocking**  (through the use of .NET managed API only) |
| Push-style API | **No** | **Yes**  [OnMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.queueclient.onmessage#Microsoft_ServiceBus_Messaging_QueueClient_OnMessage_System_Action_Microsoft_ServiceBus_Messaging_BrokeredMessage__) and **OnMessage** sessions .NET API. |
| Receive mode | **Peek & Lease** | **Peek & Lock**  **Receive & Delete** |
| Exclusive access mode | **Lease-based** | **Lock-based** |
| Lease/Lock duration | **30 seconds (default)**  **7 days (maximum)** (You can renew or release a message lease using the [UpdateMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.windowsazure.storage.queue.cloudqueue.updatemessage) API.) | **60 seconds (default)**  You can renew a message lock using the [RenewLock](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.renewlock" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_RenewLock)API. |
| Lease/Lock precision | **Message level**  (each message can have a different timeout value, which you can then update as needed while processing the message, by using the [UpdateMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.windowsazure.storage.queue.cloudqueue.updatemessage) API) | **Queue level**  (each queue has a lock precision applied to all of its messages, but you can renew the lock using the [RenewLock](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.renewlock" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_RenewLock) API.) |
| Batched receive | **Yes**  (explicitly specifying message count when retrieving messages, up to a maximum of 32 messages) | **Yes**  (implicitly enabling a pre-fetch property or explicitly through the use of transactions) |
| Batched send | **No** | **Yes**  (through the use of transactions or client-side batching) |

**Additional information**

* Messages in Storage queues are typically first-in-first-out, but sometimes they can be out of order; for example, when a message's visibility timeout duration expires (for example, as a result of a client application crashing during processing). When the visibility timeout expires, the message becomes visible again on the queue for another worker to dequeue it. At that point, the newly visible message might be placed in the queue (to be dequeued again) after a message that was originally enqueued after it.
* The guaranteed FIFO pattern in Service Bus queues requires the use of messaging sessions. In the event that the application crashes while processing a message received in the **Peek & Lock** mode, the next time a queue receiver accepts a messaging session, it will start with the failed message after its time-to-live (TTL) period expires.
* Storage queues are designed to support standard queuing scenarios, such as decoupling application components to increase scalability and tolerance for failures, load leveling, and building process workflows.
* Service Bus queues support the *At-Least-Once* delivery guarantee. In addition, the *At-Most-Once* semantic can be supported by using session state to store the application state and by using transactions to atomically receive messages and update the session state.
* Storage queues provide a uniform and consistent programming model across queues, tables, and BLOBs – both for developers and for operations teams.
* Service Bus queues provide support for local transactions in the context of a single queue.
* The **Receive and Delete** mode supported by Service Bus provides the ability to reduce the messaging operation count (and associated cost) in exchange for lowered delivery assurance.
* Storage queues provide leases with the ability to extend the leases for messages. This allows the workers to maintain short leases on messages. Thus, if a worker crashes, the message can be quickly processed again by another worker. In addition, a worker can extend the lease on a message if it needs to process it longer than the current lease time.
* Storage queues offer a visibility timeout that you can set upon the enqueuing or dequeuing of a message. In addition, you can update a message with different lease values at run-time, and update different values across messages in the same queue. Service Bus lock timeouts are defined in the queue metadata; however, you can renew the lock by calling the [RenewLock](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.renewlock" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_RenewLock)method.
* The maximum timeout for a blocking receive operation in Service Bus queues is 24 days. However, REST-based timeouts have a maximum value of 55 seconds.
* Client-side batching provided by Service Bus enables a queue client to batch multiple messages into a single send operation. Batching is only available for asynchronous send operations.
* Features such as the 200 TB ceiling of Storage queues (more when you virtualize accounts) and unlimited queues make it an ideal platform for SaaS providers.
* Storage queues provide a flexible and performant delegated access control mechanism.

**Advanced capabilities**

This section compares advanced capabilities provided by Storage queues and Service Bus queues.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Scheduled delivery | **Yes** | **Yes** |
| Automatic dead lettering | **No** | **Yes** |
| Increasing queue time-to-live value | **Yes**  (via in-place update of visibility timeout) | **Yes**  (provided via a dedicated API function) |
| Poison message support | **Yes** | **Yes** |
| In-place update | **Yes** | **Yes** |
| Server-side transaction log | **Yes** | **No** |
| Storage metrics | **Yes**  **Minute Metrics**: provides real-time metrics for availability, TPS, API call counts, error counts, and more, all in real time (aggregated per minute and reported within a few minutes from what just happened in production. For more information, see [About Storage Analytics Metrics](https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/About-Storage-Analytics-Metrics). | **Yes**  (bulk queries by calling [GetQueues](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.namespacemanager.getqueues" \l "Microsoft_ServiceBus_NamespaceManager_GetQueues)) |
| State management | **No** | **Yes**  [Microsoft.ServiceBus.Messaging.EntityStatus.Active](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.entitystatus), [Microsoft.ServiceBus.Messaging.EntityStatus.Disabled](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.entitystatus), [Microsoft.ServiceBus.Messaging.EntityStatus.SendDisabled](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.entitystatus), [Microsoft.ServiceBus.Messaging.EntityStatus.ReceiveDisabled](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.entitystatus) |
| Message auto-forwarding | **No** | **Yes** |
| Purge queue function | **Yes** | **No** |
| Message groups | **No** | **Yes**  (through the use of messaging sessions) |
| Application state per message group | **No** | **Yes** |
| Duplicate detection | **No** | **Yes**  (configurable on the sender side) |
| Browsing message groups | **No** | **Yes** |
| Fetching message sessions by ID | **No** | **Yes** |

**Additional information**

* Both queuing technologies enable a message to be scheduled for delivery at a later time.
* Queue auto-forwarding enables thousands of queues to auto-forward their messages to a single queue, from which the receiving application consumes the message. You can use this mechanism to achieve security, control flow, and isolate storage between each message publisher.
* Storage queues provide support for updating message content. You can use this functionality for persisting state information and incremental progress updates into the message so that it can be processed from the last known checkpoint, instead of starting from scratch. With Service Bus queues, you can enable the same scenario through the use of message sessions. Sessions enable you to save and retrieve the application processing state (by using [SetState](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.messagesession.setstate" \l "Microsoft_ServiceBus_Messaging_MessageSession_SetState_System_IO_Stream_) and [GetState](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.messagesession.getstate" \l "Microsoft_ServiceBus_Messaging_MessageSession_GetState)).
* [Dead lettering](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dead-letter-queues), which is only supported by Service Bus queues, can be useful for isolating messages that cannot be processed successfully by the receiving application or when messages cannot reach their destination due to an expired time-to-live (TTL) property. The TTL value specifies how long a message remains in the queue. With Service Bus, the message will be moved to a special queue called $DeadLetterQueue when the TTL period expires.
* To find "poison" messages in Storage queues, when dequeuing a message the application examines the [DequeueCount](https://msdn.microsoft.com/library/azure/microsoft.windowsazure.storage.queue.cloudqueuemessage.dequeuecount.aspx)property of the message. If **DequeueCount** is greater than a given threshold, the application moves the message to an application-defined "dead letter" queue.
* Storage queues enable you to obtain a detailed log of all of the transactions executed against the queue, as well as aggregated metrics. Both of these options are useful for debugging and understanding how your application uses Storage queues. They are also useful for performance-tuning your application and reducing the costs of using queues.
* The concept of "message sessions" supported by Service Bus enables messages that belong to a certain logical group to be associated with a given receiver, which in turn creates a session-like affinity between messages and their respective receivers. You can enable this advanced functionality in Service Bus by setting the [SessionID](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.sessionid" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_SessionId) property on a message. Receivers can then listen on a specific session ID and receive messages that share the specified session identifier.
* The duplication detection functionality supported by Service Bus queues automatically removes duplicate messages sent to a queue or topic, based on the value of the [MessageId](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.messageid" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_MessageId) property.

**Capacity and quotas**

This section compares Storage queues and Service Bus queues from the perspective of [capacity and quotas](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quotas) that may apply.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Maximum queue size | **500 TB**  (limited to a [single storage account capacity](https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction#queue-storage)) | **1 GB to 80 GB**  (defined upon creation of a queue and [enabling partitioning](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-partitioning) – see the “Additional Information” section) |
| Maximum message size | **64 KB**  (48 KB when using **Base64** encoding)  Azure supports large messages by combining queues and blobs – at which point you can enqueue up to 200 GB for a single item. | **256 KB** or **1 MB**  (including both header and body, maximum header size: 64 KB).  Depends on the [service tier](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-premium-messaging). |
| Maximum message TTL | **Infinite** (as of api-version 2017-07-27) | **TimeSpan.Max** |
| Maximum number of queues | **Unlimited** | **10,000**  (per service namespace) |
| Maximum number of concurrent clients | **Unlimited** | **Unlimited**  (100 concurrent connection limit only applies to TCP protocol-based communication) |

**Additional information**

* Service Bus enforces queue size limits. ***The maximum queue size is specified upon creation of the queue and can have a value between 1 and 80 GB***. If the queue size value set on creation of the queue is reached, additional incoming messages will be rejected and an exception will be received by the calling code. For more information about quotas in Service Bus, see [Service Bus Quotas](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quotas).
* Partitioning is not supported in the [Premium tier](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-premium-messaging). In the Standard tier, you can create Service Bus queues in 1, 2, 3, 4, or 5 GB sizes (the default is 1 GB). In Standard tier, with partitioning enabled (which is the default), Service Bus creates 16 partitions for each GB you specify. As such, if you create a queue that is 5 GB in size, with 16 partitions the maximum queue size becomes (5 \* 16) = 80 GB. You can see the maximum size of your partitioned queue or topic by looking at its entry on the [Azure portal](https://portal.azure.com/).
* With Storage queues, if the content of the message is not XML-safe, then it must be **Base64** encoded. If you **Base64**-encode the message, the user payload can be up to 48 KB, instead of 64 KB.
* With Service Bus queues, each message stored in a queue is composed of two parts: a header and a body. The total size of the message cannot exceed the maximum message size supported by the service tier.
* When clients communicate with Service Bus queues over the TCP protocol, the maximum number of concurrent connections to a single Service Bus queue is limited to 100. This number is shared between senders and receivers. If this quota is reached, subsequent requests for additional connections will be rejected and an exception will be received by the calling code. This limit is not imposed on clients connecting to the queues using REST-based API.
* If you require more than 10,000 queues in a single Service Bus namespace, you can contact the Azure support team and request an increase. To scale beyond 10,000 queues with Service Bus, you can also create additional namespaces using the [Azure portal](https://portal.azure.com/).

**Management and operations**

This section compares the management features provided by Storage queues and Service Bus queues.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Management protocol | **REST over HTTP/HTTPS** | **REST over HTTPS** |
| Runtime protocol | **REST over HTTP/HTTPS** | **REST over HTTPS**  **AMQP 1.0 Standard (TCP with TLS)** |
| .NET API | **Yes**  (.NET Storage Client API) | **Yes**  (.NET Service Bus API) |
| Native C++ | **Yes** | **Yes** |
| Java API | **Yes** | **Yes** |
| PHP API | **Yes** | **Yes** |
| Node.js API | **Yes** | **Yes** |
| Arbitrary metadata support | **Yes** | **No** |
| Queue naming rules | **Up to 63 characters long**  (Letters in a queue name must be lowercase.) | **Up to 260 characters long**  (Queue paths and names are case-insensitive.) |
| Get queue length function | **Yes**  (Approximate value if messages expire beyond the TTL without being deleted.) | **Yes**  (Exact, point-in-time value.) |
| Peek function | **Yes** | **Yes** |

**Additional information**

* Storage queues provide support for arbitrary attributes that can be applied to the queue description, in the form of name/value pairs.
* Both queue technologies offer the ability to peek a message without having to lock it, which can be useful when implementing a queue explorer/browser tool.
* The Service Bus .NET brokered messaging APIs leverage full-duplex TCP connections for improved performance when compared to REST over HTTP, and they support the AMQP 1.0 standard protocol.
* Names of Storage queues can be 3-63 characters long, can contain lowercase letters, numbers, and hyphens. For more information, see [Naming Queues and Metadata](https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/Naming-Queues-and-Metadata).
* Service Bus queue names can be up to 260 characters long and have less restrictive naming rules. Service Bus queue names can contain letters, numbers, periods, hyphens, and underscores.

**Authentication and authorization**

This section discusses the authentication and authorization features supported by Storage queues and Service Bus queues.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Authentication | **Symmetric key** | **Symmetric key** |
| Security model | Delegated access via SAS tokens. | SAS |
| Identity provider federation | **No** | **Yes** |

**Additional information**

* Every request to either of the queuing technologies must be authenticated. Public queues with anonymous access are not supported. Using [SAS](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-sas), you can address this scenario by publishing a write-only SAS, read-only SAS, or even a full-access SAS.
* The authentication scheme provided by Storage queues involves the use of a symmetric key, which is a hash-based Message Authentication Code (HMAC), computed with the SHA-256 algorithm and encoded as a **Base64** string. For more information about the respective protocol, see [Authentication for the Azure Storage Services](https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/Authentication-for-the-Azure-Storage-Services). Service Bus queues support a similar model using symmetric keys. For more information, see [Shared Access Signature Authentication with Service Bus](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-sas).

**Conclusion**

By gaining a deeper understanding of the two technologies, you will be able to make a more informed decision on which queue technology to use, and when. The decision on when to use Storage queues or Service Bus queues clearly depends on a number of factors. These factors may depend heavily on the individual needs of your application and its architecture. If your application already uses the core capabilities of Microsoft Azure, you may prefer to choose Storage queues, especially if you require basic communication and messaging between services or need queues that can be larger than 80 GB in size.

Because Service Bus queues provide a number of advanced features, such as sessions, transactions, duplicate detection, automatic dead-lettering, and durable publish/subscribe capabilities, they may be a preferred choice if you are building a hybrid application or if your application otherwise requires these features.

**Next steps**

The following articles provide more guidance and information about using Storage queues or Service Bus queues.

* [Get started with Service Bus queues](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues)
* [How to Use the Queue Storage Service](https://docs.microsoft.com/en-us/azure/storage/queues/storage-dotnet-how-to-use-queues)
* [Best practices for performance improvements using Service Bus brokered messaging](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-performance-improvements)
* [Introducing Queues and Topics in Azure Service Bus (blog post)](http://www.code-magazine.com/article.aspx?quickid=1112041)
* [The Developer's Guide to Service Bus](http://www.cloudcasts.net/devguide/Default.aspx?id=11030)
* [Using the Queuing Service in Azure](http://www.developerfusion.com/article/120197/using-the-queuing-service-in-windows-azure/)