Microsoft Integration & Cloud Architect (http://microsoftintegration.guru/)

Cloud & Integration Specialist based in UK

HOME (HTTP://MICROSOFTINTEGRATION.GURU/)
BLOG (HTTP://MICROSOFTINTEGRATION.GURU/BLOG/)
PUBLICATIONS (HTTP://MICROSOFTINTEGRATION.GURU/PUBLICATIONS/)
SPEAKING (HTTP://MICROSOFTINTEGRATION.GURU/SPEAKING/)
COMMUNITY (HTTP://MICROSOFTINTEGRATION.GURU/COMMUNITY/)
AWARDS (HTTP://MICROSOFTINTEGRATION.GURU/AWARDS/)
RESUME (HTTP://MICROSOFTINTEGRATION.GURU/RESUME/)
CONTACT US (HTTP://MICROSOFTINTEGRATION.GURU/CONTACT/)

Azure Event Hubs vs Azure Messaging

March 3, 2015 (http://microsoftintegration.guru/2015/03/03/azure-event-hubs-vs-azure-messaging/) / michaelstephensonuk (http://microsoftintegration.guru/author/michaelstephensonuk/)

Recently Ive been chatting with a few people about Azure Service Bus and it's clear that in the community there is some confusion about the differences between Azure Service Bus Messaging (queues and topics) and Azure Service Bus Event Hubs and where you should use each. I thought i'd share my thoughts around this.

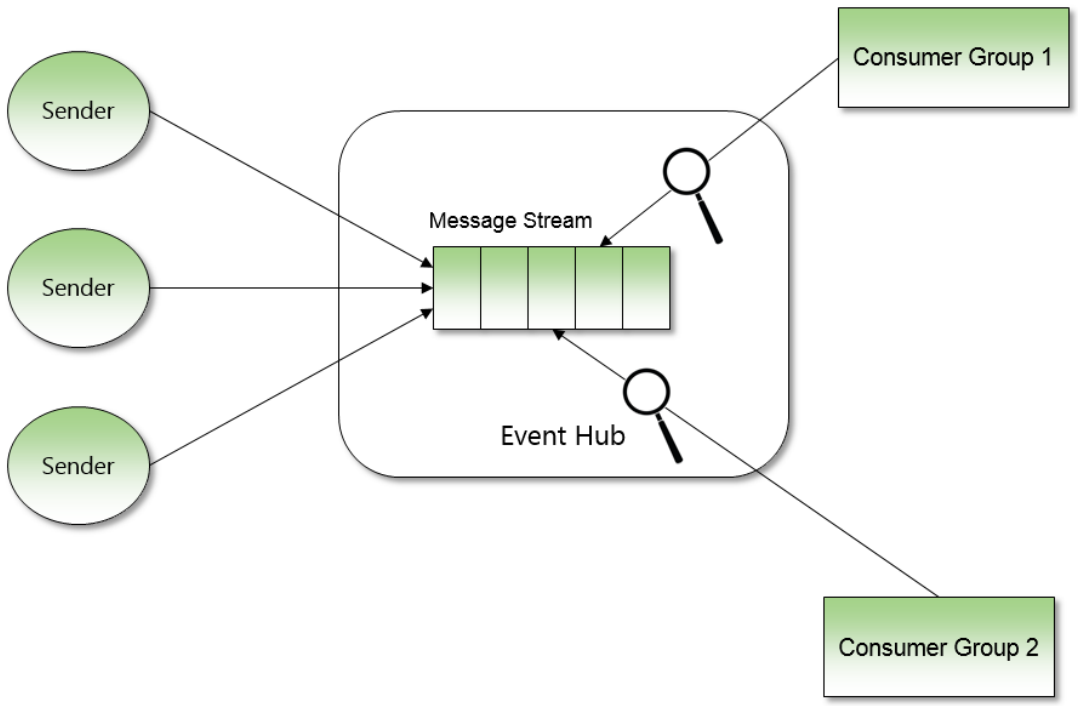
What is Azure Service Bus Event Hubs

Azure Service Bus Event Hubs is a relatively new feature within the Azure Service Bus and is intended to help with the challenge of handling an event based messaging at huge scale. The idea is that if you have apps or devices publishing telemetry events then Event Hubs can be the place you would send them to and behind the scenes the Event Hub will create a stream of all of these events which can be read at some point in different ways.

Event Hubs provides simple interfaces such as AMQP and HTTP to make it easy for apps to send messages to an Event Hub. Internally Event Hubs implements a partitioning pattern to allow it to scale to deal with huge bursts of messages and to retain messages for a longer period of time.

In Event Hubs you can define consumer groups which allow you to read the stream of events. If you only need one receiver to read the stream then you can use the default consumer group, but if you need multiple receivers to read the stream concurrently but at their own rate then each receiver would use its own consumer group. A receiver will also manage an index (or off set) which is its own pointer to where in the stream of messages it is reading. A receiver can start at the beginning of the stream and read to the end and then wait for new events or alternatively it can start reading part way through the stream.

The below diagram shows what Event Hubs might look like:



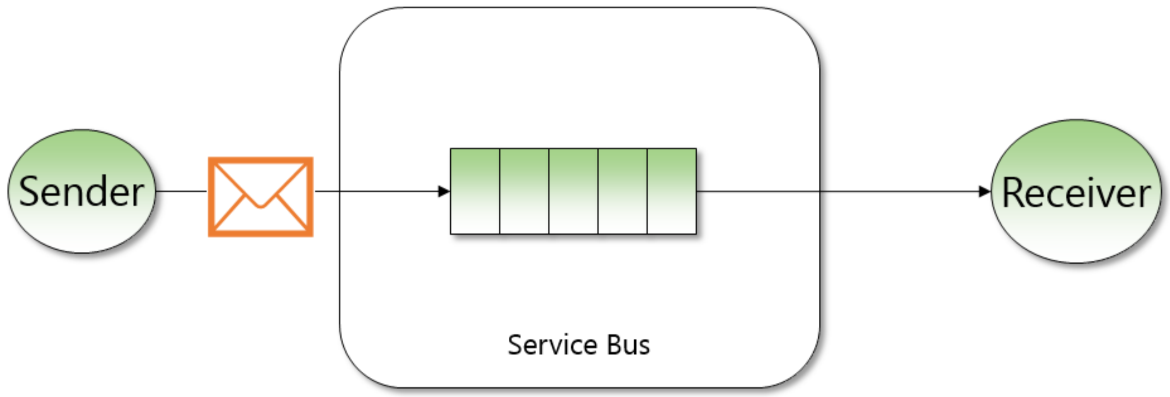
(http://microsoftintegration.guru/wp-content/uploads/2015/03/Hub.png)

What is Azure Service Bus Messaging

Azure Service Bus messaging is an implementation of modern message queuing concepts implemented in the Microsoft Cloud as platform as a service. Service Bus messaging has two key areas:

- Queues
- Topics

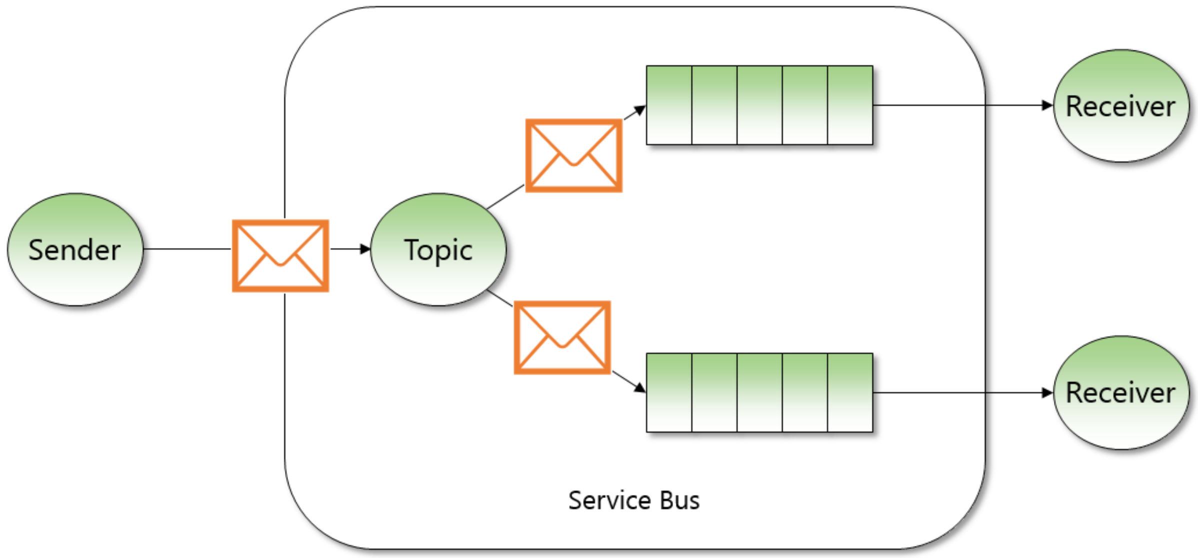
A queue implements a one way communication where the sender places a message on the queue and a receiver will collect the message soon or sometime in the future. The queue is a durable entity meaning that a message placed on a queue is saved and can remain on the queue for a long period of time if the receiver is not ready to collect it yet. A simple diagram showing a queue concept is below:



(http://microsoftintegration.guru/wp-content/uploads/2015/03/queue.png)

A topic is similar to a queue except that a message sent to a topic can have a copy of that message forwarded to multiple subscriptions. A subscription is similar to a queue except that is includes some rules which determine which messages from a topic will go to each queue. An example of the use of a topic could be to use a fan out pattern where all messages sent to a topic will have a copy sent to each subscription so many applications can get their own copy of all messages. Another example would be the implementation of a publish and subscribe pattern where the message sent to a topic is only forwarded to a subset of queues based on the rules it matches. An example would be if a message had a property called “Message Type” and a subscription could say only send messages where the message type is “CustomerUpdate” to the subscription used by the CRM application.

The below diagram is a simple representation of the topic pattern.



(http://microsoftintegration.guru/wp-content/uploads/2015/03/image2015-2-24-23-52-38.png)

Similarities

To the external publisher of an event/message or the receiver or an event/message Service Bus Messaging and Service Bus Event Hubs can look very similar and this is what makes it difficult to understand the differences between the two and where to use what.

The main similarity is that both messaging and event hubs can be interacted with via AMQP so they look conceptually like you're sending a message to a queue like entity. Other similarities include:

- Both entities live within an Azure Service Bus Namespace and can be managed in that part of the Azure Portal.
- Both entities can also support an HTTP interface
- Both entities have durability features so that a message can be persisted and received later
- Both entities support a kind of time to live style feature but they are not necessarily the same
- Both options support the same SAS security model by default for sending, receiving and management
- Both options have the same 256kb message size limitation

Differences

While Service Bus Event Hubs and Service Bus Messaging may look similar(ish) to a sender and receiver there are some significant differences in the internal workings of the features, the use cases then are intended to support and the specifics of how each feature works.

Use Cases

From a target use case perspective if we consider some of our typical enterprise integration patterns then if you are implementing a pattern which uses a Command Message (http://www.eaipatterns.com/CommandMessage.html), or a Request/Reply Message (http://www.eaipatterns.com/RequestReply.html) then you probably want to use Azure Service Bus Messaging. RPC patterns can be implemented using Request/Reply messages on Azure Service Bus using a response queue. These are really about ESB and EAI style messaging patterns where you want to send messages between applications and probably want to use other features such as property based routing.

Azure Event Hubs is more likely to be used if you're implementing patterns with Event Messages (http://www.eaipatterns.com/EventMessage.html) and you want somewhere reliable to send them that is capable of dealing with a massive scale but will allow you to do stuff with the events out of process.

With these core target use cases in mind it is easy to see where the scale differences come into play. For messaging it's about one application telling one or more apps to DO SOMETHING or GIVE ME SOMETHING. The alternative is that in eventing the applications are saying SOMETHING HAS HAPPENED. When you consider this in typical application scenarios and you put events into the telemetry and logging space you can quickly see that the SOMETHING HAS HAPPENED scenario will produce a lot more traffic than the other.

Now I'm not saying that you can't implement some messaging type functions using event hubs and that you can't push events to a Service Bus topic as in integration there are always different requirements which result in different implementation scenarios, but I think if you follow the above as a general rule then you will usually be on the right path.

I guess one additional point about Event Hubs is that some of the most likely use cases for Event Hubs would involve combining the hub with some other Azure feature which will do interesting things with the telemetry events it receives. As an example I would expect one of the most common use cases would be using Stream Analytics and Machine Learning and Power BI to implement reporting solutions on top of aggregates of the events which are received. For these use cases Stream Analytics has its own Event Hub connector so it can process messages from the hub. Likewise in Azure Service Bus Messaging products like BizTalk Server and BizTalk Services have out of the box connectors for queues and topics which allow you to do typical EAI/ESB integration patterns by combining Service Bus + BizTalk.

Internal Workings

I mentioned earlier that the inner workings of Messaging and Event Hubs are different and that is predominantly around the typical use case requirements and supporting the scale you would want for those use cases. If you consider that in Messaging you have concepts like Property Based Routing, Dead Letter Queues, which provide some rich features for flowing messages between applications and managing who gets which message and handling error scenarios. In Events Hub many of these rich features have been sacrificed to allow the event platform to scale to be able to handle a much larger volume of messages. By sacrificing some of these features it is assumed that they are not needed for the key use cases of event processing or would be implemented in the receiver.

An example of this could be that in Service Bus Messaging a receiver can read a message from a queue and then it will be deleted once successfully processed. In Event Hubs the receiver does not remove messages from the stream and instead would manage its own index or check point to know where its position in the stream is. Using this approach allows the Event Hub to have a single stream of messages that all consumers can read. This is different to Messaging where each consumer will get their own copy of the message which at Event Hubs scale could cause a problem when receiving a million messages per second results in you having to create 10 million messages so every receiver has their own copy. It's pretty obvious to see that the Topic pattern has limits as to how far it could scale.

Costs

While the cost models are very similar Event Hubs is optimized for those very high scale usage scenarios. As you scale up your usage, you could expect a cost comparison to tip towards Event Hubs being the lower cost. As an example you can process 2.6 billion 1KB events per month for around $97 on Event Hubs. In messaging you would need to buy blocks of message operations that unless using Basic it would likely be more expensive. Each message would require 3 operations.

I guess at some point I might do some like for like comparisons to flush out more details on this cost difference but for now you could expect at lower scale there probably isnt a great deal difference in the cost but at very high usage you would notice a difference.

Lower Level Features

If you dig down in to the lower level aspects of the two Service Bus features there are some other differences that you should be aware of.

Time to Live and Retention

In Service Bus messaging we are used to the idea that a message can live on a queue for as long as the queue exists if no one receives it. We are also used to the option to have each message have its own time to live which will mean that when a message "expires" it will be moved to the dead letter queue. Event Hubs has a slightly different concept of this. There is no specific time to live for an event but there is a retention policy on the hub. This is set as a period of time in days and by default means events will remain on the hub for 7 days and then automatically be removed. You can increase this up to a limit. Obviously the longer the retention the more messages are likely to remain in the stream.

FIFO

With Azure Service Bus Messaging it is possible to implement first in – first out style patterns if you use a queue and a single process receiving the messages.

In Event Hubs it is perhaps less likely you would try to implement a FIFO pattern with a hub but it can be done. You could achieve this pattern if you use the publisher key so that all events with the same key are sent to the same partition and they would then be stored in the order which they were received. This could be a useful pattern for things like localization of data and keeping a stream of events next to each other so a down stream processor doesnt need to read them from multiple places or worry about thread locking.

With FIFO this can be achieved with both Service Bus features, but the way it would be implemented is a bit different and you would need to make different considerations for each way. As an example with Event Hubs you would be sacrificing a lot of the scale potential offered by partitions by processing messages in order and that is probably not a typical use case for Event Hubs, but it can still be done.

Related Messages

In Azure Service Bus Messaging there is the concept of a message session. This allows you to indicate that a number of messages are related because they have the same session id property. This allows you to implement some rich integration patterns such as a splitter and aggregator. You can have groups of related messages within different sessions on the same queue. Event Hubs does not have the same concept of a message session but you can relate messages by putting them on the same partition. In this case its just a different kind of relationship.

ACS Support

Azure Service Bus Messaging still provides support for security tokens obtained from ACS. Event Hubs does not offer this.

Although this is a difference you should look to move away from using ACS with Service Bus Messaging anyway.

Receiver Differences

I mentioned earlier in this post about the difference between the receive patterns around a queue's peek/lock/delete option and the idea that a message is removed from a queue when processed vs the Event Hubs idea that the receiver does not remove messages from the stream and instead manages its own pointer to where it is reading in the queue. There are a few other differences which are discussed below.

Competing Consumers

In Service Bus Messaging you can scale up the receivers of a message by using the Competing Consumer pattern (http://www.eaipatterns.com/CompetingConsumers.html). This means you can scale out by having many processes all pointing to the same queue and each would take turns in getting a message. This can support you scaling out to quite a high level and you can also implement patterns where topics are used to forward messages to multiple queues so you can split the messages to scale even higher.

In Event Hubs it works slightly differently and does not really work on a competing consumer pattern. To scale out your load of messages you would increase the number of partitions and then you would have a single processor for each partition. This should let you have a high throughput of messages by having many partitions. You can also benefit from not having to acknowledge a message you simply read it, often in batches so you will get super high throughput.

The tradeoff for this is it is more difficult to support high availability on your processor for each partition. The best way to do this is to use the EventProcessorHost class which is discussed in this article (http://blogs.msdn.com/b/servicebus/archive/2015/01/16/event-processor-host-best-practices-part-1.aspx) which provides a set of classes which take care of the hard work for you. Fortunately the check point pattern will allow your recovered host to get back to where it was in the stream but I guess we may see more to come in terms of guidance around this in future articles from the product team.

Message Replay

In much of this article you are probably getting the picture that Event Hubs is sacrificing features which Messaging has to be able to achieve higher scale for use cases which don't need those features which were sacrificed. One of the features that event hubs has which Service Bus Messaging does not is the ability to replay messages which have already been processed. This is a really cool concept and is something you get because the messages were not removed from the stream. You can simply change the index the receiver is looking for to be at any point in the stream and it will start processing from there and read all messages again. You could rewind back to the start of the stream or only go back one message.

Reject & Abandon Message

In Service Bus Messaging you may get a message which could not be processed. An example is you may receive a message from a queue for a customer who does not exist. You may choose to ignore the message or you could choose to reject it by sending it to the dead letter queue or finally if you thought the message could not be processed for now but a retry might work, you could abandon the message so it is released back to the queue and another receiver would pick it up again. This gives you some useful features where Service Bus Messaging will help you deal with messages which do not process successfully. This can be used to handle temporary errors, poison messages and malformed messages.

In Event Hubs it's really the receivers problem to deal with those scenarios. You would read an event message from the stream and then you do something and move on to the next index or alternatively you handle the error and move on to the next message. You cannot for example remove a malformed event message from the stream so if you rewind and read this again you will need to handle the dodgy event a 2nd time.

Other Differences

A few other differences to consider are listed below.

On Premise Support

Azure Service Bus Messaging is also available for on premise installations as part of the Azure Pack for Windows Server. Event Hubs is not available as an on premise offering at the time of writing and in my opinion for the use cases targeted by event hubs most customers are unlikely to want to have the scale of infrastructure that is probably required which probably makes Event Hubs one of those features which would be a lower priority to offer on premise.

Summary

Hopefully this provides lots of information for you to consider when thinking about a decision whether you should use Azure Service Bus Messaging or Azure Service Bus Event Hubs. As I mentioned earlier in the post for many people I think if you follow the guide of the target use cases you wont go too far wrong and hopefully if you have one of those scenarios where you dont fall clearly into either camp or you have conflicting requirements which make the decision not straight forward then the above info will provide some lower level considerations to help you choose the right option for your scenario.

Acknowledgements

A quick thanks to Kent Weare and Dan Rosanova for reviewing the above for me.

G+1 2 Like 19 Tweet 23

Azure (http://microsoftintegration.guru/category/azure/), Service Bus (http://microsoftintegration.guru/category/service-bus/) / permalink (http://microsoftintegration.guru/2015/03/03/azure-event-hubs-vs-azure-messaging/)

← DEV BOX TOOLS (HTTP://MICROSOFTINTEGRATION.GURU/2015/02/09/DEV-BOX-TOOLS/)

LONDON BIZTALK SUMMIT- AN INTEGRATION PLATFORM TO SUPPORT VISION 2025 → (HTTP://MICROSOFTINTEGRATION.GURU/2015/03/05/LONDON-BIZTALK-SUMMIT-INTEGRATION-PLATFORM-SUPPORT-VISION-2025/)

4 Comments My Blog 1 Login

Recommend 2 Share Sort by Best

Join the discussion…

monkeysupport • 5 months ago
The guys over at Microsoft need to learn how to write like you. Well written mate.
2 • Reply • Share ›

michael stephenson monkeysupport • 5 months ago
thank you appreciate the compliment
• Reply • Share ›

Muhammad Asif Ashraf • 2 months ago
I exactly want to repeat it "The guys over at Microsoft need to learn how to write like you. Well written mate."
• Reply • Share ›

Rıfat Erdem Sahin • 4 months ago
great post
• Reply • Share ›

ALSO ON **MY BLOG** WHAT'S THIS?

Whats in your integration platform?
1 comment • 9 months ago
Phil Chalk — After the first 8 months of having any sort of structured integration, we now have: Mule ESB (on premise) …

BizTalk Server vNext (2016) – What 6 things would you like to see?
7 comments • 5 months ago
sqlRune — Thanks for this article. I administrate a small solution (Biztalk 2009) and have reached the max 5 apps. I am …

BizTalk Generic Streamed Download Pipeline Approach
1 comment • 9 months ago
Akshay Shaha — Thanks for this article... :)

Why use Service Bus Relay now I have Hybrid Connections?
1 comment • a year ago
Gyanendra Gautam — Nice comparison and uses of both the technologies.

Subscribe Add Disqus to your site Privacy

The Azure Coach

Zero to Cloud in 1 Day

CODING KNOWLEDGE *not* REQUIRED

(http://theazurecoach.com/courses/zero-to-cloud-in-1-day/)

MVP Microsoft® Most Valuable Professional

SOCIAL MEDIA

(https://www.linkedin.com/in/michaelstephe

(https://twitter.com/michael_stephen)

(http://mvp.microsoft.com/en-us/mvp/Michael%20Stephenson-4021792)

CONTACT INFO

Email:
michael@connectedsystemsconsulting.co.uk

Tel: +44 7971 912 353

RECENT POSTS

› BizTalk Server + Event Hubs (http://microsoftintegration.guru/2015/09/06/biztalk-server-event-hubs/)
› Azure Training Workshop Competition (http://microsoftintegration.guru/2015/09/04/azure-training-workshop-competition/)
› Integration Monday – Who is Sriram? (http://microsoftintegration.guru/2015/09/04/integration-monday-who-is-sriram/)
› How do you do analysis for a BizTalk map? (http://microsoftintegration.guru/2015/08/21/how-do-you-do-analysis-for-a-biztalk-map/)
› Deployment Notifications to Yammer – via Console App (http://microsoftintegration.guru/2015/06/12/deployment-notifications-to-yammer-via-console-app/)

CATEGORIES

› API (http://microsoftintegration.guru/category/api/)
› API App (http://microsoftintegration.guru/category/api-app/)
› Azure (http://microsoftintegration.guru/category/azure/)
› Azure App Service

(http://microsoftintegration.guru/category/azure-app-service/)
- BizTalk (http://microsoftintegration.guru/category/biztalk/)
- CRM (http://microsoftintegration.guru/category/crm/)
- Logic App (http://microsoftintegration.guru/category/logic-app/)
- RabbitMQ (http://microsoftintegration.guru/category/rabbitmq/)
- Service Bus (http://microsoftintegration.guru/category/service-bus/)
- Uncategorized (http://microsoftintegration.guru/category/uncategorized/)

ARCHIVES

- September 2015 (http://microsoftintegration.guru/2015/09/)
- August 2015 (http://microsoftintegration.guru/2015/08/)
- June 2015 (http://microsoftintegration.guru/2015/06/)
- May 2015 (http://microsoftintegration.guru/2015/05/)
- April 2015 (http://microsoftintegration.guru/2015/04/)
- March 2015 (http://microsoftintegration.guru/2015/03/)
- February 2015 (http://microsoftintegration.guru/2015/02/)
- January 2015 (http://microsoftintegration.guru/2015/01/)
- December 2014 (http://microsoftintegration.guru/2014/12/)
- November 2014

(http://microsoftintegration.guru/2014/11/)
› October 2014 (http://microsoftintegration.guru/2014/10/)
› July 2014 (http://microsoftintegration.guru/2014/07/)
› June 2014 (http://microsoftintegration.guru/2014/06/)

(HTTP://GEEKSWITHBLOGS.NET/MICHAELSTEPHENSON/RSS.ASPX) MY BLOG (HTTP://GEEKSWITHBLOGS.NET/MICHAELSTEPHENSON/DEFAULT.ASPX)

› Goals for Integration Projects (http://geekswithblogs.net/michaelstephenson/archive/2014/10/16/159731.aspx)
› API Management Video (http://geekswithblogs.net/michaelstephenson/archive/2014/08/03/157900.aspx)
› Hybrid Connections Webcast (http://geekswithblogs.net/michaelstephenson/archive/2014/07/08/157411.aspx)
› Why use Service Bus Relay when I can use Hybrid Connections? (http://geekswithblogs.net/michaelstephenson/archive/2014/07/07/157350.aspx)
› 8 ways any BizTalk customer can use Azure right now (http://geekswithblogs.net/michaelstephenson/archive/2014/06/05/156805.aspx)
› BizTalk Server Best Practices (http://geekswithblogs.net/michaelstephenson/archive/2014/05/19/156438.aspx)
› BizTalk HL7 Testing - Tool to get config from BTS Management DB (http://geekswithblogs.net/michaelstephenson/archive/2014/04/21/156030.aspx)
› Automated Testing of BizTalk HL7 solutions with Specflow (http://geekswithblogs.net/michaelstephenson/archive/2014/04/20/156021.aspx)
› BizTalk Services – Can I create a mapping service in the cloud (http://geekswithblogs.net/michaelstephenson/archive/2014/03/17/155701.aspx)
› Real-world WABS - Part 1 (http://geekswithblogs.net/michaelstephenson/archive/2014/03/09/155627.aspx)

Proudly powered by WordPress (http://wordpress.org/) | Theme: Solon (http://athemes.com/theme/solon) by aThemes