Table of Contents

Service Bus messaging: flexible data delivery in the cloud

1/17/2017 • 4 min to read • Edit on GitHub

Microsoft Azure Service Bus is a reliable information delivery service. The purpose of this service is to make communication easier. When two or more parties want to exchange information, they need a communication mechanism. Service Bus is a brokered, or third-party communication mechanism. This is similar to a postal service in the physical world. Postal services make it very easy to send different kinds of letters and packages with a variety of delivery guarantees, anywhere in the world.

Similar to the postal service delivering letters, Service Bus is flexible information delivery from both the sender and the recipient. The messaging service ensures that the information is delivered even if the two parties are never both online at the same time, or if they aren't available at the exact same time. In this way, messaging is similar to sending a letter, while non-brokered communication is similar to placing a phone call (or how a phone call used to be - before call waiting and caller ID, which are much more like brokered messaging).

The message sender can also require a variety of delivery characteristics including transactions, duplicate detection, time-based expiration, and batching. These patterns have postal analogies as well: repeat delivery, required signature, address change, or recall.

Service Bus supports two distinct messaging patterns: *Relay* and *brokered messaging*.

Service Bus Relay

The WCF Relay component of Service Bus is a centralized (but highly load-balanced) service that supports a variety of different transport protocols and Web services standards. This includes SOAP, WS-*, and even REST. The relay service provides a variety of different relay connectivity options and can help negotiate direct peer-to-peer connections when it is possible. Service Bus is optimized for .NET developers who use the Windows Communication Foundation (WCF), both with regard to performance and usability, and provides full access to its relay service through SOAP and REST interfaces. This makes it possible for any SOAP or REST programming environment to integrate with Service Bus.

The relay service supports traditional one-way messaging, request/response messaging, and peer-to-peer messaging. It also supports event distribution at Internet-scope to enable publish-subscribe scenarios and bi-directional socket communication for increased point-to-point efficiency. In the relayed messaging pattern, an on-premises service connects to the relay service through an outbound port and creates a bi-directional socket for communication tied to a particular rendezvous address. The client can then communicate with the on-premises service by sending messages to the relay service targeting the rendezvous address. The relay service will then "relay" messages to the on-premises service through the bi-directional socket already in place. The client does not need a direct connection to the on-premises service, nor is it required to know where the service resides, and the on-premises service does not need any inbound ports open on the firewall.

You initiate the connection between your on-premises service and the relay service, using a suite of WCF "relay" bindings. Behind the scenes, the relay bindings map to transport binding elements designed to create WCF channel components that integrate with Service Bus in the cloud.

Service Bus WCF Relay provides many benefits, but requires the server and client to both be online at the same time in order to send and receive messages. This is not optimal for HTTP-style communication, in which the requests may not be typically long-lived, nor for clients that connect only occasionally, such as browsers, mobile applications, and so on. Brokered messaging supports decoupled communication, and has its own advantages;

clients and servers can connect when needed and perform their operations in an asynchronous manner.

Brokered messaging

In contrast to the relay scheme, brokered messaging can be thought of as asynchronous, or "temporally decoupled." Producers (senders) and consumers (receivers) do not have to be online at the same time. The messaging infrastructure reliably stores messages in a "broker" (such as a queue) until the consuming party is ready to receive them. This allows the components of the distributed application to be disconnected, either voluntarily; for example, for maintenance, or due to a component crash, without affecting the entire system. Furthermore, the receiving application may only have to come online during certain times of the day, such as an inventory management system that only is required to run at the end of the business day.

The core components of the Service Bus brokered messaging infrastructure are queues, topics, and subscriptions. The primary difference is that topics support publish/subscribe capabilities that can be used for sophisticated content-based routing and delivery logic, including sending to multiple recipients. These components enable new asynchronous messaging scenarios, such as temporal decoupling, publish/subscribe, and load balancing. For more information about these messaging entities, see Service Bus queues, topics, and subscriptions.

As with the WCF Relay infrastructure, the brokered messaging capability is provided for WCF and .NET Framework programmers, and also via REST.

Next steps

To learn more about Service Bus messaging, see the following topics.

- Service Bus fundamentals
- Service Bus queues, topics, and subscriptions
- How to use Service Bus queues
- How to use Service Bus topics and subscriptions

What is Azure Relay?

1/17/2017 • 2 min to read • Edit on GitHub

The Azure Relay service facilitates hybrid applications by enabling you to securely expose services that reside within a corporate enterprise network to the public cloud, without having to open a firewall connection, or require intrusive changes to a corporate network infrastructure. Relay supports a variety of different transport protocols and web services standards.

The relay service supports traditional one-way, request/response, and peer-to-peer traffic. It also supports event distribution at internet-scope to enable publish/subscribe scenarios and bi-directional socket communication for increased point-to-point efficiency.

In the relayed data transfer pattern, an on-premises service connects to the relay service through an outbound port and creates a bi-directional socket for communication tied to a particular rendezvous address. The client can then communicate with the on-premises service by sending traffic to the relay service targeting the rendezvous address. The relay service will then "relay" data to the on-prem service through a bi-directional socket dedicated to each client. The client does not need a direct connection to the on-premises service, it is not required to know where the service resides, and the on-premises service does not need any inbound ports open on the firewall.

The key capability elements provided by Relay are bi-directional, unbuffered communication across network boundaries with TCP-like throttling, endpoint discovery, connectivity status, and overlaid endpoint security. The relay capabilities differ from network-level integration technologies such as VPN, in that relay can be scoped to a single application endpoint on a single machine, while VPN technology is far more intrusive as it relies on altering the network environment.

Azure Relay has two features:

1. Hybrid Connections - Uses the open standard web sockets enabling multi-platform scenarios.
2. WCF Relays - Uses Windows Communication Foundation (WCF) to enable remote procedure calls. WCF Relay is the legacy Relay offering that many customers may already use with their WCF programming models.

Hybrid Connections and WCF Relays both enable secure connection to assets that exist within a corporate enterprise network. Use of one over the other is dependent on your particular needs, as described in the following table:

	WCF RELAY	HYBRID CONNECTIONS
WCF	x	
.NET Core		x
.NET Framework	x	x
JavaScript/NodeJS*		x
Java*		x
Standards-Based Open Protocol		x
Multiple RPC Programming Models		x

*By General Availability

Hybrid Connections

The Azure Relay Hybrid Connections capability is a secure, open-protocol evolution of the existing Relay features that can be implemented on any platform and in any language that has a basic WebSocket capability, which explicitly includes the WebSocket API in common web browsers. Hybrid Connections is based on HTTP and WebSockets.

WCF Relays

The WCF Relay works for the full .NET Framework (NETFX) and for WCF. You initiate the connection between your on-premises service and the relay service using a suite of WCF "relay" bindings. Behind the scenes, the relay bindings map to new transport binding elements designed to create WCF channel components that integrate with Service Bus in the cloud.

Service history

Hybrid Connections supplants the former, similarly named "BizTalk Services" feature that was built on the Azure Service Bus WCF Relay. The new Hybrid Connections capability complements the existing WCF Relay feature and these two service capabilities exist side-by-side in the Relay service for the foreseeable future. They share a common gateway, but are otherwise different implementations.

Next steps:

- Relay FAQ
- Create a namespace
- Get started with .NET
- Get started with Node