Microsoft IoT Camp #5

# Introduce Microsoft Azure

김영욱 Technical Evangelist
부장/ DX / Microsoft

youngwook@outlook.com
Blog: Youngwook.com

# Azure Service Bus

## 서비스 버스 기본 사항

각 상황에 따라 다른 스타일의 통신이 요청됩니다. 때로는 응용 프로그램이 단순한 큐를 통해 메시지를 보내고 받도록 하는 것이 최상의 솔루션입니다. 다른 상황에서는 일반적인 큐로 충분하지 않고 게시 및 구독 메커니즘을 사용한 큐가 더 효율적입니다. 실제로 응용 프로그램 간의 연결만 필요하고 큐가 필요하지 않은 경우도 있습니다. 서비스 버스는 세 가지 옵션을 모두 제공하여 응용 프로그램이 여러 다른 방법으로 상호 작용할 수 있게 해 줍니다.

서비스 버스는 다중 테넌트 클라우드 서비스로, 여러 사용자가 서비스를 공유합니다. 응용 프로그램 개발자 등의 각 사용자는 *namespace*를 만든 후 해당 네임스페이스에서 필요한 통신 메커니즘을 정의합니다. [그림 1]에서는(#Fig1) 표시되는 모양을 보여 줍니다.

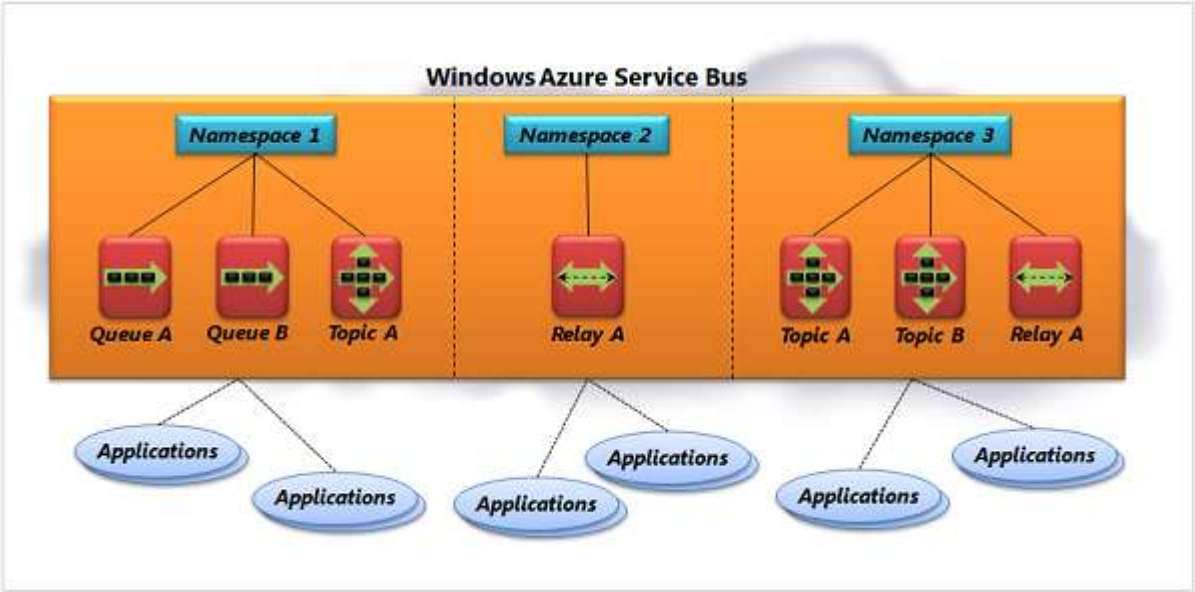



**그림 1: 서비스 버스는 클라우드를 통해 응용 프로그램을 연결하기 위한 다중 테넌트 서비스를 제공합니다.**

네임스페이스 내에서 각각 다른 방식으로 응용 프로그램을 연결하는 세 가지 통신 메커니즘 인스턴스를 하나 이상 사용할 수 있습니다. 선택 항목은 다음과 같습니다.

- *Queues* - 단방향 통신을 허용합니다. 각 큐는 수신될 때까지 전송된 메시지를 저장하는 중간자( *broker*라고도 함) 역할을 합니다.
- *Topics* - *subscriptions*을 사용하여 단방향 통신을 제공합니다. 큐와 마찬가지로 토픽은 브로커 역할을 하지만 각 구독에서 특정 기준과 일치하는 메시지만 표시할 수 있게 해 줍니다.
- *Relays* - 양방향 통신을 제공합니다. 큐 및 토픽과 달리 릴레이는 처리 중인 메시지를 저장하지 않으며 브로커가 아닙니다. 단순히 메시지를 대상 응용 프로그램으로 전달합니다.

큐, 토픽 또는 릴레이를 만들 때 이름을 지정합니다. 이 이름은 네임스페이스 이름과 결합되어 개체의 고유 식별자를 만듭니다. 응용 프로그램은 서비스 버스에 이 이름을 제공한 다음 해당 큐, 토픽 또는 릴레이를 사용하여 서로 통신할 수 있습니다.

이러한 개체를 사용하기 위해 Windows 응용 프로그램은 WCF(Windows Communication Foundation)를 사용할 수 있습니다. 큐와 토픽의 경우 Windows 응용 프로그램이 서비스 버스에서 정의된 메시지 API 를 사용할 수도 있습니다. HTTP 를 통해 큐와 토픽에 액세스할 수도 있으며, Microsoft 는 Windows 가 아닌 응용 프로그램에서 큐와 토픽을 사용하기 쉽도록 Java, Node.js 및 기타 언어용 SDK 를 제공합니다.

서비스 버스 자체가 클라우드(즉, Microsoft Azure 데이터 센터)에서 실행되는 경우에도 서비스 버스를 사용하는 응용 프로그램은 다른 곳에서 실행될 수 있음을 이해하는 것이 중요합니다. 예를 들어 서비스 버스를 사용하여 Azure 에서 실행되는 응용 프로그램이나 고유한 데이터 센터 내부에서 실행되는 응용 프로그램을 연결할 수 있습니다. 서비스 버스를 사용하여 Azure 또는 다른 클라우드 플랫폼에서 실행되는 응용 프로그램을 온-프레미스 응용 프로그램이나 태블릿 및 휴대폰과 연결할 수도 있습니다. 가전 제품, 센서 및 기타 장치를 중앙 응용 프로그램에 연결하거나 서로 연결할 수도 있습니다. 서비스 버스는 거의

모든 곳에서 액세스할 수 있는 클라우드의 일반 통신 메커니즘입니다. 사용 방법은 응용 프로그램에서 수행해야 하는 작업에 따라 달라집니다.

## 큐

서비스 버스 큐를 사용하여 두 개의 응용 프로그램을 연결한다고 가정해 보세요. [그림 2]에서는(#Fig2) 이 상황을 보여 줍니다.

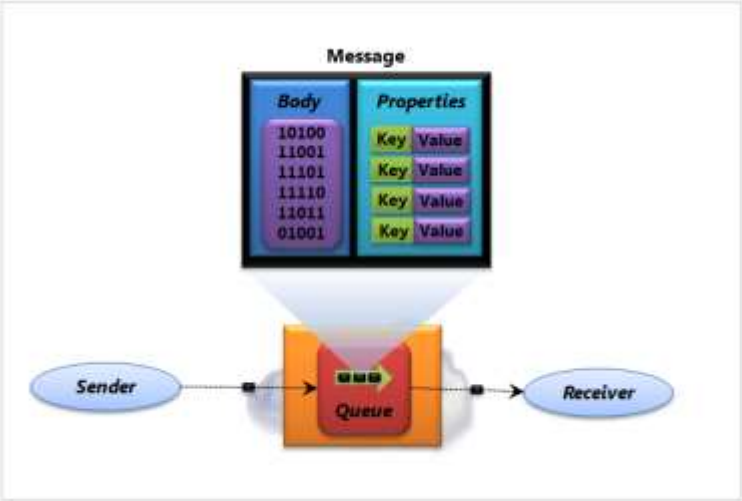



**그림 2: 서비스 버스 큐는 단방향 비동기 큐를 제공합니다.**

프로세스는 간단합니다. 센서가 메시지를 서비스 버스 큐로 보내면 수신기가 나중에 해당 메시지를 확인합니다. 각 큐에 하나의 수신기만 있거나(그림 2 참조) 여러 응용 프로그램이 동일한 큐에서 읽을 수 있습니다. 두 번째 상황에서는 일반적으로 각 메시지가 하나의 수신기에서만 읽혀지며 큐에서 멀티캐스트 서비스를 제공하지 않습니다.

각 메시지는 각각 키/값 쌍인 속성 집합과 이진 메시지 본문의 두 부분으로 이루어져 있습니다. 메시지가 사용되는 방법은 응용 프로그램에서 수행하려는 작업에 따라 달라집니다. 예를 들어 최근 판매에 대한 메시지를 보내는 응용 프로그램은 *Seller="Ava"* 및 *Amount=10000* 속성을 포함할 수 있습니다. 메시지 본문은 서명된 판매 계약의 스캔 이미지를 포함하거나, 이러한 이미지가 없는 경우 비어 있을 수 있습니다.

수신기는 두 가지 방법으로 서비스 버스 큐에서 메시지를 읽을 수 있습니다. 첫 번째 옵션은 ReceiveAndDelete 라고 하며, 큐에서 메시지를 제거하고 즉시 삭제합니다. 이 옵션은 간단하지만 수신기에서 메시지 처리를 마치기 전에 크래시가 발생할 경우 메시지가 손실됩니다. 큐에서 제거되었기 때문에 다른 수신기가 메시지에 액세스할 수 없습니다.

두 번째 옵션은 PeekLock 이라고 하며, 이 문제를 해결하는 데 도움이 됩니다. ReceiveAndDelete 와 마찬가지로 PeekLock 읽기는 큐에서 메시지를 제거합니다. 그러나 메시지를 삭제하지 않습니다. 대신, 메시지를 잠가서 다른 수신기에 표시되지 않도록 하고 다음 세 가지 이벤트 중 하나를 기다립니다.

- 수신기가 메시지 처리에 성공하고 Complete 를 호출하면 큐가 메시지를 삭제합니다.

- 수신기가 메시지를 처리할 수 없다고 결정하고 Abandon 을 호출하면 큐가 메시지에서 잠금을 제거하고 다른 수신기가 사용할 수 있게 합니다.
- 수신기가 구성 가능한 기간(기본적으로 60 초) 내에 둘 다 호출하지 않으면 큐가 수신기에서 실패했다고 가정합니다. 이 경우 수신기가 Abandon 을 호출한 것처럼 동작하고 다른 수신기가 메시지를 사용할 수 있게 합니다.

여기서 발생할 수 있는 문제는 동일한 메시지가 두 개의 수신기에 두 번 배달될 수 있다는 것입니다. 서비스 버스 큐를 사용하는 응용 프로그램은 이 문제에 대비해야 합니다. 중복 검색이 용이하도록 각 메시지에는 고유한 MessageID 속성이 있습니다. 기본적으로 이 속성은 큐에서 메시지를 읽는 횟수에 관계없이 동일하게 유지됩니다.

큐는 다양한 상황에서 유용합니다. 큐를 사용하면 응용 프로그램이 동시에 실행되지 않는 경우에도 서로 통신할 수 있으므로 일괄 처리 및 모바일 응용 프로그램에서 특히 유용합니다. 여러 수신기가 있는 큐는 전송된 메시지가 이러한 수신기에 분산되므로 자동 부하 분산 기능도 제공합니다.

## 토픽

유용하긴 하지만 큐가 항상 올바른 솔루션인 것은 아닙니다. 때로는 서비스 버스 토픽이 더 효율적입니다. [그림 3]에서는(#Fig3) 이 아이디어를 보여줍니다.

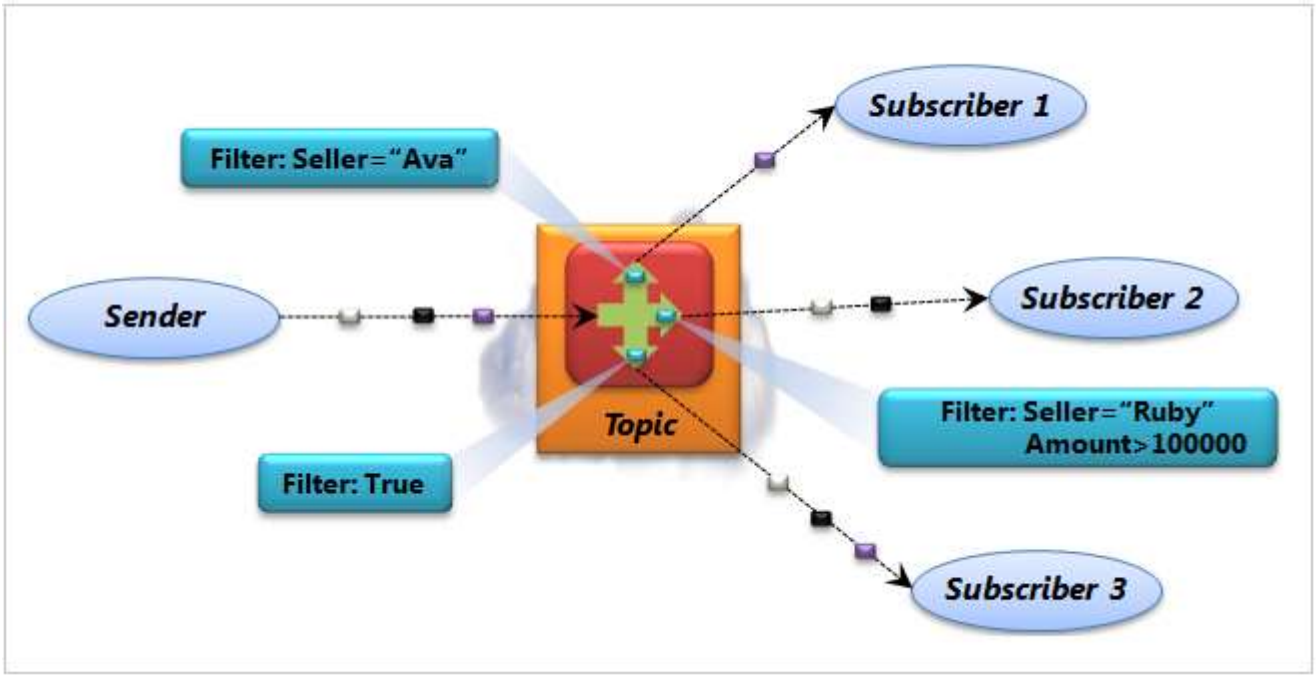



**그림 3: 구독 응용 프로그램에서 지정한 필터를 기준으로 서비스 버스 토픽에 전송된 메시지를 일부 또는 모두 받을 수 있습니다.**

토픽은 여러 측면에서 큐와 유사합니다. 보낸 사람은 메시지를 큐에 제출하는 것과 동일한 방식으로 메시지를 토픽에 제출하며, 이러한 메시지는 큐와 동일하게 표시됩니다. 큰 차이점은 토픽을 사용할 경우 각

수신 응용 프로그램이 *filter*를 정의하여 고유한 구독을 만들 수 있다는 것입니다. 그러면 구독자가 해당 필터와 일치하는 메시지만 볼 수 있습니다. 예를 들어 그림 3 에서는 각각 고유한 필터가 있는 세 명의 구독자가 포함된 토픽과 보낸 사람을 보여 줍니다.

- 구독자 1 은 *Seller="Ava"* 속성이 포함된 메시지만 받습니다.
- 구독자 2 는 *Seller="Ruby"* 속성과 값이 100,000 보다 큰 *Amount* 속성이 포함된 메시지를 받습니다. Ruby 는 판매 관리자이므로 자신의 매출과 판매자에 관계없이 모든 대규모 매출을 보려고 합니다.
- 구독자 3 은 해당 필터를 *True*로 설정합니다. 이렇게 하면 모든 메시지를 받습니다. 예를 들어 이 응용 프로그램에서 감사 내역을 유지 관리하므로 모든 메시지를 확인해야 할 수도 있습니다.

큐와 마찬가지로 토픽 구독자는 ReceiveAndDelete 또는 PeekLock 을 사용하여 메시지를 읽을 수 있습니다. 그러나 큐와 달리 토픽으로 전송된 단일 메시지를 여러 구독자가 받을 수 있습니다. 흔히 *publish and subscribe*이라고 불리는 이 접근 방법은 여러 응용 프로그램이 동일한 메시지에 관련된 경우에 유용합니다. 올바른 필터를 정의하면 각 구독자가 확인해야 하는 메시지 스트림 부분만 볼 수 있습니다.

## 릴레이

큐와 토픽은 둘 다 브로커를 통해 단방향 비동기 통신을 제공합니다. 트래픽이 한 방향으로만 진행되며, 보낸 사람과 받는 사람 간에 직접 연결이 없습니다. 그러나 이런 방식을 원하지 않는 경우 어떻게 해야 할까요? 응용 프로그램이 보내기와 받기를 모두 수행해야 하거나, 그 사이에 메시지를 저장할 장소가 필요하지 않도록 직접 연결을 사용하려 한다고 가정해 보세요. 이러한 문제를 처리하기 위해 서비스 버스는 그림 4 와 같은 릴레이를 제공합니다.

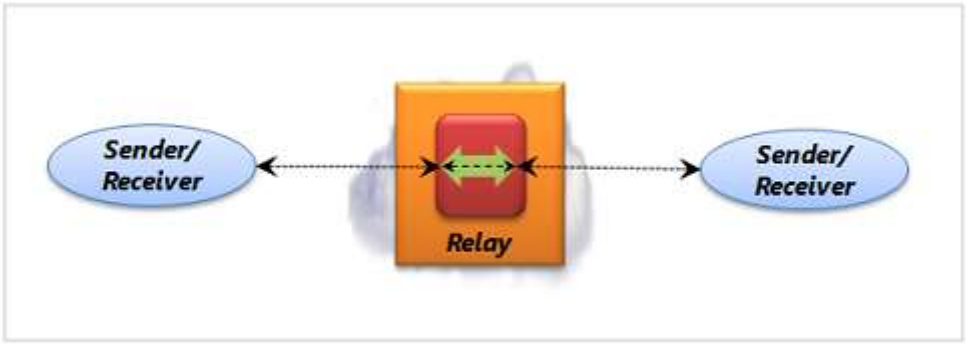



**그림 4: 서비스 버스 릴레이는 응용 프로그램 간에 양방향 동기 통신을 제공합니다.**

릴레이와 관련해서 제기되는 질문은 릴레이를 사용해야 하는 이유입니다. 큐가 필요하지 않다고 해도 왜 응용 프로그램이 직접 상호 작용하는 대신 클라우드 서비스를 통해 통신해야 할까요? 그 대답은 직접 통신하는 것이 생각보다 어려울 수 있기 때문입니다.

회사 데이터 센터 내부에서 실행되는 두 개의 온-프레미스 응용 프로그램을 연결하려 한다고 가정해 보세요. 각 응용 프로그램은 방화벽 뒤에 있고, 각 데이터 센터에서 NAT(Network Address Translation)를 사용합니다. 방화벽은 몇 개의 포트를 제외한 모든 포트에서 들어오는 데이터를 차단하며, NAT 는 각 응용 프로그램이

실행되는 컴퓨터에 고정 IP 주소가 없음을 암시합니다. 추가 도움이 없으면 공용 인터넷을 통해 이러한 응용 프로그램을 연결하는 데 문제가 있습니다.

서비스 버스 릴레이는 이 도움을 제공합니다. 릴레이를 통해 양방향으로 통신하기 위해 각 응용 프로그램은 서비스 버스와 아웃바운드 TCP 연결을 설정한 다음 열어 둡니다. 두 응용 프로그램 간의 모든 통신은 이러한 연결을 통해 전송됩니다. 각 연결이 데이터 센터 내부에서 설정되었으므로 새 포트를 열지 않고도 방화벽에서 들어오는 트래픽, 즉 "릴레이를 통해 전송된 데이터"를 각 응용 프로그램으로 허용합니다. 이 접근 방식을 사용할 경우 통신 전체 과정에서 각 응용 프로그램에 일관된 끝점이 있으므로 NAT 문제도 해결됩니다. 릴레이를 통해 데이터를 교환하면 응용 프로그램에서 통신을 어렵게 만드는 문제를 피할 수 있습니다.

서비스 버스 릴레이를 사용하기 위해 응용 프로그램은 WCF(Windows Communication Foundation)를 사용합니다. 서비스 버스는 Windows 응용 프로그램이 릴레이를 통해 간단하게 상호 작용할 수 있게 해 주는 WCF 바인딩을 제공합니다. 이미 WCF 를 사용하는 응용 프로그램은 일반적으로 이러한 바인딩 중 하나만 지정한 다음 릴레이를 통해 서로 통신할 수 있습니다. 그러나 큐 및 토픽과 달리 Windows 가 아닌 응용 프로그램에서 릴레이를 사용할 수는 있지만 이 경우 약간의 프로그래밍이 필요하며 표준 라이브러리가 제공되지 않습니다.

큐 및 토픽과 달리 응용 프로그램에서 명시적으로 릴레이를 만들지는 않습니다. 대신, 메시지를 받으려는 응용 프로그램이 서비스 버스와 TCP 연결을 설정하면 릴레이가 자동으로 만들어집니다. 연결을 삭제하면 릴레이가 삭제됩니다. 응용 프로그램이 특정 수신기에서 만든 릴레이를 찾을 수 있도록 서비스 버스는 이름으로 특정 릴레이를 찾을 수 있게 해 주는 레지스트리를 제공합니다.

릴레이는 직접 통신이 필요한 경우에 올바른 솔루션입니다. 예를 들어 온-프레미스 데이터 센터에서 실행되는 항공 예약 시스템을 생각해 보세요. 이 시스템은 체크 인 키오스크, 모바일 장치 및 기타 컴퓨터에서 액세스해야 합니다. 어디서 실행되든 관계없이 이러한 모든 시스템에서 실행되는 응용 프로그램은 클라우드의 서비스 버스 릴레이를 사용하여 통신할 수 있습니다.

응용 프로그램 연결은 항상 전체 솔루션 빌드의 일부였으며, 이 문제가 완전히 사라지기는 어렵습니다. 이 큐, 토픽 및 릴레이를 통해 이를 수행하기 위한 클라우드 기반 기술을 제공함으로써 서비스 버스는 이 필수 기능을 더 쉽고 보다 광범위하게 사용할 수 있게 하려고 합니다.

## 이벤트 허브 개요

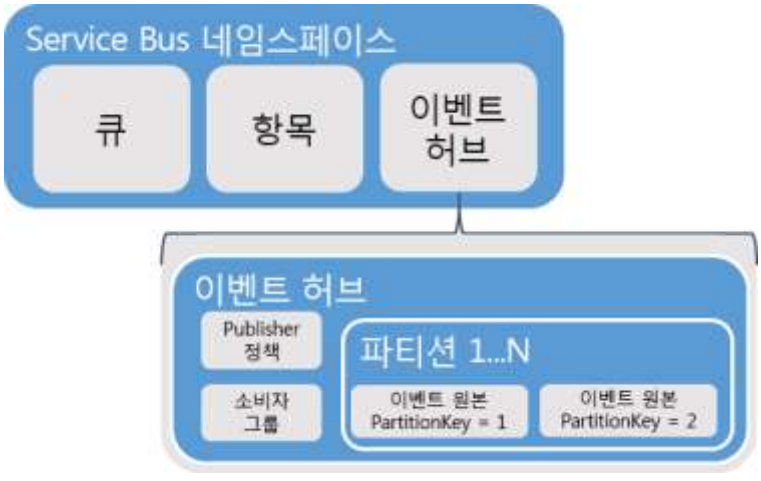
**https://msdn.microsoft.com/ko-kr/library/azure/dn789973.aspx**

적응형 고객 환경을 제공하거나 지속적인 피드백 및 자동화된 원격 분석을 통해 제품을 개선하고자 하는 대부분의 최신 솔루션에는 다수의 동시 게시자가 제공하는 매우 많은 양의 정보를 안정적이며 안전하게 수집하는 방법이 필요합니다. Microsoft Azure 이벤트 허브는 폭넓은 시나리오에서 대량의 데이터 수집을 위한 기반을 제공하는 관리되는 플랫폼 서비스입니다. 이러한 시나리오의 예로는 모바일 앱의 동작 추적, 웹 팜의 트래픽 정보, 콘솔 게임의 게임 내 이벤트 캡처, 산업 기계나 연결된 차량에서 수집되는 원격 분석 데이터 등이 있습니다. 이러한 솔루션 아키텍처에서 이벤트 허브는 공통적으로 이벤트 파이프라인의

"출입구", 즉 *이벤트 수집기* 역할을 합니다. 이벤트 수집기는 이벤트 생산자와 이벤트 소비자 사이에서 이벤트 스트림 생산과 이벤트 사용을 분리하는 구성 요소 또는 서비스입니다.


Microsoft Azure 서비스 버스 이벤트 허브는 짧은 대기 시간과 우수한 안정성으로 클라우드에 대규모 이벤트 및 원격 분석을 수신하는 기능을 제공하는 이벤트 수집기 서비스입니다. 이 서비스를 다른 다운스트림 서비스와 함께 사용할 경우 응용 프로그램 계측, 사용자 환경 또는 워크플로 처리 및 IoT(사물 인터넷) 시나리오에서 특히 유용합니다. 이벤트 허브는 메시지 스트림 처리 기능을 제공하며, 큐 및 항목과 비슷한 엔터티이기는 하지만 기존의 엔터프라이즈 메시징과는 특성이 매우 다릅니다. 엔터프라이즈 메시징 시나리오에서는 보통 시퀀싱, 배달 못 한 편지 처리, 트랜잭션 지원, 강력한 배달 보증 등의 여러 가지 복잡한 기능이 필요하지만 이벤트 수집에서 중요한 기능은 이벤트 스트림을 위한 높은 처리량 및 처리 유연성입니다. 따라서 Azure 이벤트 허브 기능은 주로 높은 처리량 및 이벤트 처리 기능이 필요한 시나리오에 사용된다는 점에서 Service Bus 항목과는 다릅니다. 그러므로 이벤트 허브는 항목에 대해 사용 가능한 일부 메시징 기능을 구현하지 않습니다. 이러한 기능이 필요한 경우에는 항목을 선택하는 것이 좋습니다.

이벤트 허브는 큐 및 항목과 비슷하게 Service Bus 의 네임스페이스 수준에서 작성되며, 기본 API 인터페이스로 AMQP 및 HTTP 를 사용합니다. 아래 다이어그램에는 이벤트 허브가 Service Bus 에서 사용되는 방식이 나와 있습니다.

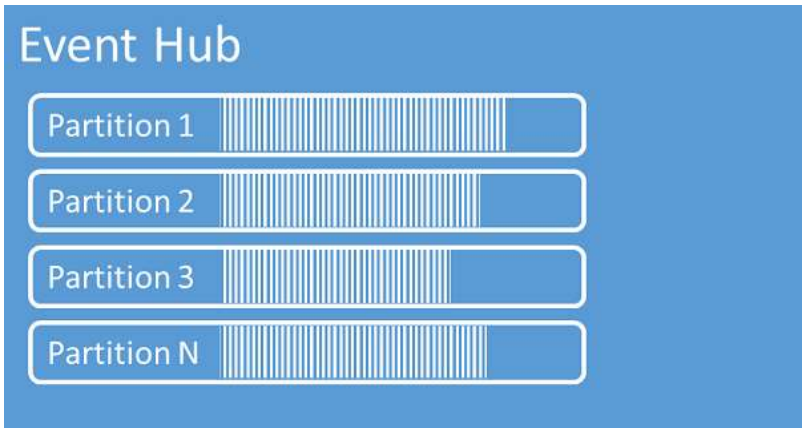



이벤트 허브는 분할된 소비자 패턴을 통해 메시지 스트리밍 기능을 제공합니다. 큐와 항목은 각 소비자가 같은 큐나 리소스에서 읽기를 시도하는 경합하는 소비자 모델을 사용합니다. 이러한 리소스 경합으로 인해 스트림 처리 응용 프로그램이 복잡해지고 규모가 제한됩니다. 반면 이벤트 허브는 각 소비자가 메시지 스트림의 특정 하위 집합, 즉 파티션만을 읽는 분할된 소비자 패턴을 사용합니다. 이 패턴에서는 이벤트 처리를 위해 매우 폭넓은 수평 확장이 가능하며, 큐와 항목에서는 사용할 수 없는 기타 스트림 중심 기능이 제공됩니다.

### 파티션

파티션은 이벤트 허브에서 발생하는 이벤트의 순서가 지정된 시퀀스입니다. 새로 도착한 이벤트는 이 시퀀스의 끝에 추가됩니다. 파티션은 "커밋 로그"로 간주할 수 있습니다.


파티션은 이벤트 허브 수준에서 설정되며 이벤트 허브의 모든 파티션에 적용되는 구성된 보존 시간 동안 데이터를 보존합니다. 이벤트는 시간별로 만료되며 명시적으로 삭제할 수는 없습니다. 이벤트 허브에는 여러 파티션이 포함됩니다. 각 파티션은 독립적이며 고유한 데이터 시퀀스를 포함합니다. 그러므로 파티션은 각기 다른 속도로 확장됩니다.




파티션 수는 이벤트 허브를 만들 때 지정하며 8~32 개 사이여야 합니다. 기본적으로는 파티션을 사용하는 응용 프로그램에서 필요한 다운스트림 병렬 처리의 수준에 따라 파티션의 수가 결정됩니다. 파티션은 데이터 구성 메커니즘이며 이벤트 허브 처리량보다는 다운스트림 병렬 처리와 보다 관련성이 높습니다. 따라서 이벤트 허브에서 선택하는 파티션의 수는 예상되는 동시 독자의 수와 직접적으로 관련됩니다. 이벤트 허브를 만든 후에는 파티션 수를 변경할 수 없으므로 장기적으로 예상되는 병렬 처리 측면에서 이 수를 고려해야 합니다. Microsoft Azure 서비스 버스 팀에 문의하여 파티션 제한인 32 개를 늘릴 수 있습니다.

파티션은 식별 가능하며 데이터를 직접 보낼 수 있지만 일반적으로는 특정 파티션으로 데이터를 보내지 않는 것이 좋습니다. 대신 "이벤트 게시자" 및 "게시자 정책" 섹션에서 소개하는 더 높은 수준의 구문을 사용할 수 있습니다.

### 이벤트 데이터

이벤트 허브 컨텍스트에서는 메시지를 *이벤트 데이터*라고 합니다. 이벤트 데이터는 이벤트 본문, 사용자 정의 속성 모음, 그리고 이벤트에 대한 여러 메타데이터(예: 파티션의 이벤트 오프셋, 스트림 시퀀스의 이벤트 수)를 포함합니다. 파티션에는 이벤트 데이터 시퀀스가 채워집니다.

### 이벤트 게시자

이벤트 허브로 이벤트 또는 데이터를 전송하는 모든 엔터티는 *이벤트 게시자*입니다. 이벤트 게시자는 HTTPS 또는 AMQP 1.0 을 사용하여 이벤트를 게시할 수 있습니다. 이벤트 게시자는 SAS(공유 액세스 서명) 토큰을 사용하여 이벤트 허브가 자신을 식별하도록 하며, 시나리오의 요구 사항에 따라 고유한 ID 를 포함하거나 공통 SAS 토큰을 사용할 수 있습니다.

SAS 사용에 대한 자세한 내용은 Service Bus 의 공유 액세스 서명 인증을 참조하세요.

### 일반 게시자 태스크

이 섹션에서는 이벤트 게시자의 일반적인 태스크에 대해 설명합니다.

#### SAS 토큰 얻기

SAS(공유 액세스 서명)는 이벤트 허브의 인증 메커니즘입니다. Service Bus 에서는 네임스페이스 및 이벤트 허브 수준에서 SAS 정책을 제공합니다. SAS 키에서 생성되는 SAS 토큰은 특정 형식으로 인코딩된 URL 의 SHA 해시입니다. Service Bus 는 토큰과 키(정책)의 이름을 사용하여 해시를 다시 생성해 발신자를 인증할 수 있습니다. 일반적으로 이벤트 게시자용 SAS 토큰은 특정 이벤트 허브에 대한 보내기 권한을 통해서만 작성됩니다. 이 SAS 토큰 URL 메커니즘은 게시자 정책에서 소개하는 게시자 식별의 기준이 됩니다. SAS 사용에 대한 자세한 내용은 **Service Bus 의 공유 액세스 서명 인증**을 참조하세요.

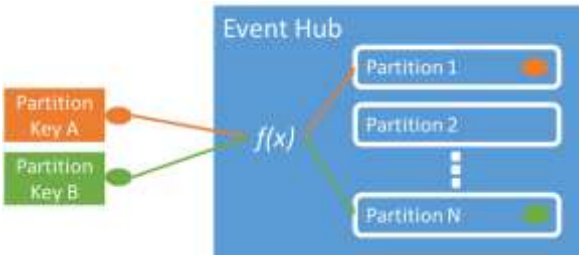
#### 이벤트 게시

AMQP 1.0 또는 HTTP 를 통해 이벤트를 게시할 수 있습니다. Service Bus 에서는 .NET 클라이언트에서 이벤트 허브에 이벤트를 게시하는 데 사용할 수 있는 EventHubClient 클래스를 제공합니다. 기타 런타임 및 플랫폼의 경우 Apache Qpid 와 같은 AMQP 1.0 클라이언트를 사용할 수 있습니다. 이벤트는 개별적으로 게시할 수도 있고 일괄로 게시할 수도 있습니다. 단일 게시(이벤트 데이터 인스턴스)의 제한은 단일 이벤트인지 배치인지에 관계없이 256KB 입니다. 이 크기보다 큰 이벤트를 게시하면 오류가 발생합니다. 게시자는 이벤트 허브 내의 파티션을 인식하지 않고 아래에서 소개하는 **PartitionKey** 만 지정하거나 SAS 토큰을 통해 ID 를 지정하는 것이 좋습니다.

사용 시나리오에 따라 AMQP 나 HTTP 중 사용할 방법을 선택합니다. AMQP 를 사용하려면 TLS(전송 계층 보안) 또는 SSL/TLS 외에 영구 양방향 소켓을 설정해야 합니다. 이 작업을 수행하려면 네트워크 트래픽 측면에서 비용이 많이 들 수 있습니다. 그러나 AMQP 세션을 시작할 때만 설정을 수행하면 됩니다. HTTP 의 경우 초기 오버헤드는 낮지만 모든 요청에 대해 추가 SSL 오버헤드가 필요합니다. 이벤트를 자주 게시하는 게시자의 경우 AMQP 를 사용하면 성능, 대기 시간 및 처리량과 관련한 비용을 크게 줄일 수 있습니다.

#### 파티션 키

파티션 키는 데이터 구성을 위해 들어오는 이벤트 데이터를 특정 파티션에 매핑하는 데 사용되는 값이며, 이벤트 허브로 전달되는 발신자가 제공한 값입니다. 이 값은 정적 해시 기능을 통해 처리되며 해당 결과를 토대로 파티션이 할당됩니다. 이벤트를 게시할 때 파티션 키를 지정하지 않으면 라운드 로빈 할당이 사용됩니다. 파티션 키를 사용하는 경우 이벤트 게시자는 파티션 키만 인식하며 이벤트가 게시되는 파티션은 인식하지 않습니다. 이와 같은 분리로 인해 발신자는 이벤트 저장 및 다운스트림 처리에 대해 자세히 파악하지 않아도 됩니다. 파티션 키는 다운스트림 처리를 위해 데이터를 구성할 때는 중요하지만 기본적으로 파티션 자체와는 관련이 없습니다. 장치 단위 ID 나 사용자의 고유 ID 를 파티션 키로 사용해도 되지만 지역 등의 기타 특성을 사용하여 관련 이벤트를 단일 파티션으로 그룹화할 수도 있습니다. 아래 이미지에는 이벤트 발신자가 파티션 키를 사용하여 각 파티션에 연결하는 방식이 나와 있습니다.




이벤트 허브를 사용하는 경우 같은 파티션 키 값을 공유하는 모든 이벤트가 같은 파티션에 순서대로 배달됩니다. 여기서 중요한 점은 다음 섹션에서 설명하는 게시자 정책과 함께 파티션 키를 사용하는 경우 게시자의 ID와 파티션 키의 값이 일치해야 한다는 것입니다. 그렇지 않으면 오류가 발생합니다.

### 이벤트 소비자

이벤트 허브에서 이벤트 데이터를 읽는 모든 엔터티는 *이벤트 소비자*입니다. 모든 이벤트 소비자는 소비자 그룹의 파티션을 통해 이벤트 스트림을 읽습니다. 각 파티션에는 활성 독자가 한 번에 하나씩만 포함되어야 합니다. 모든 이벤트 허브 소비자는 AMQP 1.0 세션을 통해 연결하며 이벤트는 사용 가능해지면 배달됩니다. 클라이언트는 데이터 가용성을 폴링하지 않아도 됩니다.

### 소비자 그룹

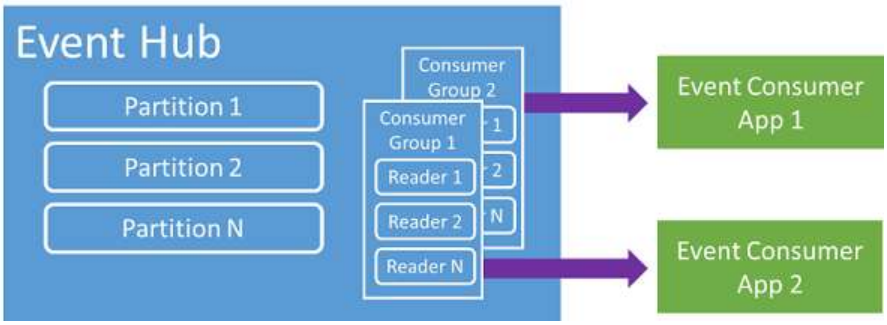
*소비자 그룹*을 통해 이벤트 허브의 게시/구독 메커니즘을 사용하도록 설정합니다. 소비자 그룹은 전체 이벤트 허브의 보기(상태, 위치 또는 오프셋)입니다. 소비자 그룹을 통해 데이터를 사용하는 여러 응용 프로그램이 각각 개별 이벤트 스트림 보기를 사용할 수 있으며 고유한 속도와 오프셋으로 스트림을 독립적으로 읽을 수 있습니다. 스트림 처리 아키텍처에서는 각 다운스트림 응용 프로그램이 소비자 그룹에 해당합니다. 장기 저장소에 이벤트 데이터를 쓰려는 경우에는 해당 저장소 작성기 응용 프로그램이 소비자 그룹입니다. 복잡한 이벤트 처리는 다른 별도의 소비자 그룹에서 수행합니다. 소비자 그룹을 통해서만 파티션에 액세스할 수 있습니다. 이벤트 허브에는 항상 기본 소비자 그룹이 있으며 표준 계층 이벤트 허브에 대해 소비자 그룹을 20 개까지 만들 수 있습니다.

아래에는 소비자 그룹 URI 규칙의 예가 나와 있습니다.

복사

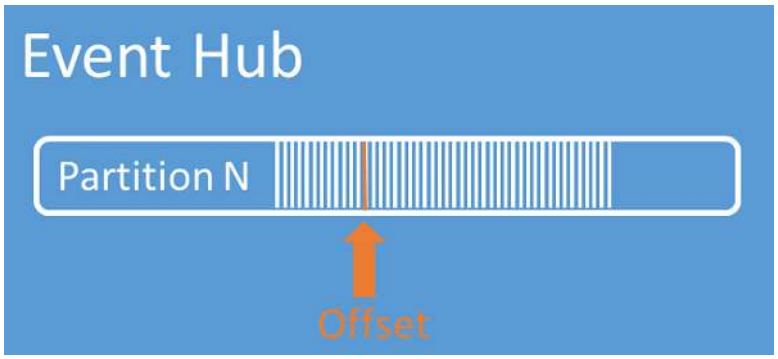
```
//<my namespace>.servicebus.windows.net/<event hub name>/<Consumer Group #1>
//<my namespace>.servicebus.windows.net/<event hub name>/<Consumer Group #2>
```

아래 이미지에는 소비자 그룹 내의 이벤트 소비자가 나와 있습니다.




### 스트림 오프셋

오프셋은 파티션 내에서 이벤트의 위치이며, 클라이언트 쪽 커서로 간주할 수 있습니다. 오프셋은 이벤트의 바이트 번호이므로 이벤트 소비자(독자)가 이벤트 읽기를 시작하려는 이벤트 스트림의 지점을 지정할 수 있습니다. 오프셋은 타임스탬프 또는 오프셋 값으로 지정할 수 있습니다. 소비자는 이벤트 허브 서비스 외부에 자체 오프셋 값을 저장해야 합니다.




파티션 내에서 각 이벤트에는 오프셋이 포함됩니다. 소비자는 이 오프셋을 사용하여 지정된 파티션의 이벤트 시퀀스 위치를 표시합니다. 독자가 연결할 때 오프셋을 타임스탬프 값 또는 번호로 이벤트 허브에 전달할 수 있습니다.

### 검사점 설정

*검사점 설정*은 독자가 파티션 이벤트 시퀀스 내에서 해당 위치를 표시하거나 커밋하는 데 사용하는 프로세스로, 소비자가 수행해야 하며 소비자 그룹 내에서 파티션 단위로 수행됩니다. 즉, 각 파티션 독자는 각 소비자 그룹에 대해 이벤트 스트림에서 현재 위치를 추적해야 하며 데이터 스트림이 완료되었다고 간주되면 서비스에 알릴 수 있습니다. 독자는 파티션에서 연결을 끊었다가 다시 연결할 때 해당 소비자 그룹 내 해당 파티션의 마지막 독자가 이전에 제출한 검사점에서 읽기를 시작합니다. 독자는 연결 시 이 오프셋을 이벤트 허브에 전달하여 읽기를 시작할 위치를 지정합니다. 이러한 방식을 통해 검사점 설정을 사용하여 이벤트가 다운스트림 응용 프로그램에 의해 "완료"된 것으로 표시할 수 있는 동시에, 각기 다른 컴퓨터에서 실행되는 독자 간의 장애 조치(failover) 시에 복원 기능을 제공할 수 있습니다. 이벤트 데이터는 이벤트 허브를 만들 때 지정한 보존 간격 동안 보존되므로 이 검사점 설정 프로세스에서 더 낮은 오프셋을 지정하여 이전 데이터를 반환할 수 있습니다. 검사점 설정에서 이러한 메커니즘을 사용하면 장애 조치(failover) 복원 및 제어되는 이벤트 스트림 재생이 가능합니다.

### 일반 소비자 태스크

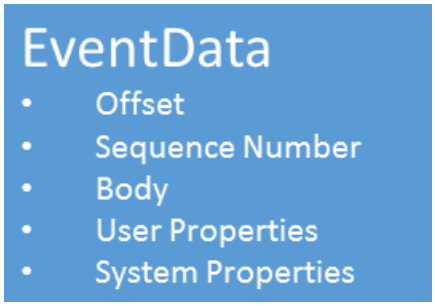
이 섹션에서는 이벤트 허브 이벤트 소비자 또는 독자가 일반적으로 수행하는 태스크에 대해 설명합니다. 모든 이벤트 허브 소비자는 AMQP 1.0 을 통해 연결합니다. AMQP 1.0 은 세션 및 상태 인식 양방향 통신 채널입니다. 각 파티션에는 파티션별로 분리된 이벤트 전송을 원활하게 수행할 수 있는 AMQP 1.0 링크 세션이 있습니다.

### 파티션에 연결

이벤트 허브에서 이벤트를 사용하려면 소비자가 파티션에 연결해야 합니다. 앞에서 설명한 것처럼 항상 소비자 그룹을 통해 파티션에 액세스합니다. 분할된 소비자 모델에서는 소비자 그룹 내의 파티션에서 한 번에 하나의 독자만 활성 상태여야 합니다. 일반적으로는 파티션에 직접 연결할 때 특정 파티션에 대한 독자 연결을 조정하기 위해 임대 메커니즘을 사용합니다. 이 방식을 사용하는 경우 소비자 그룹의 모든 파티션은 활성 독자를 하나만 포함할 수 있습니다. 시퀀스에서 독자의 위치를 관리하는 태스크는 검사점 설정을 통해 수행되는 중요한 작업입니다. .NET 클라이언트용 EventProcessorHost 클래스를 사용하면 이 기능을 간소화할 수 있습니다. 지능형 소비자 에이전트인 EventProcessorHost 에 대해서는 다음 섹션에서 설명합니다.

### 이벤트 읽기

특정 파티션에 대해 AMQP 1.0 세션 및 링크를 열면 이벤트 허브 서비스가 이벤트를 AMQP 1.0 클라이언트로 배달합니다. 이 배달 메커니즘에서는 HTTP GET 등의 끌어오기 기반 메커니즘에 비해 처리량은 높이고 대기 시간은 줄일 수 있습니다. 이벤트를 클라이언트로 전송할 때 각 이벤트 데이터 인스턴스는 이벤트 시퀀스에 대한 검사점 설정을 원활하게 수행하는 데 사용되는 오프셋, 시퀀스 번호 등의 중요 메타데이터를 포함합니다.




사용자는 스트림 처리 진행률을 가장 효율적으로 관리할 수 있는 방식으로 이 오프셋을 관리해야 합니다.

## 용량 및 보안

이벤트는 스트림 수신을 위한 확장성이 매우 뛰어난 병렬 아키텍처입니다. 따라서 이벤트 허브를 기반으로 솔루션 크기를 지정하고 확장할 때는 몇 가지 주요 측면을 고려해야 합니다. 이러한 용량 컨트롤의 첫 번째 요소가 다음 섹션에서 설명하는 *처리량 단위*입니다.

### 처리량 단위

이벤트 허브의 처리 용량은 처리량 단위를 통해 제어됩니다. 처리량 단위는 선불 방식 용량 단위입니다. 단일 처리량 단위에는 다음이 포함됩니다.

- 수신: 초당 최대 1MB/1,000 개 이벤트
- 송신: 초당 최대 2MB

수신은 구입한 처리량 단위 수를 기준으로 제공되는 용량으로 제한됩니다. 이 용량을 초과하여 데이터를 전송하면 할당량 초과 예외가 발생합니다. 이 용량은 초당 1MB 또는 1,000 개 이벤트 중 먼저 도달하는 값입니다. 송신에서는 제한 예외가 발생하지 않지만 송신 용량은 구입한 처리량 단위를 기준으로 제공되는 데이터 전송 용량인 처리량 단위당 초당 2MB 로 제한됩니다. 게시 속도 예외가 표시되거나 송신 속도를 높이려는 경우 이벤트 허브를 만든 네임스페이스에 대해 구입한 처리량 단위를 확인하세요. 처리량 단위를 높이려는 경우 Azure 포털의 **구성** 탭 **네임스페이스** 페이지에서 설정을 조정하면 됩니다. Azure API 를 통해 이 설정을 변경할 수도 있습니다.

파티션은 데이터 구성 관련 개념인 반면 처리량 단위는 순수한 용량 관련 개념입니다. 처리량 단위는 선불 방식이며, 시간 단위로 요금이 청구됩니다. 처리량 단위 구입 후에는 최소 1 시간에 해당하는 요금이 청구됩니다. Service Bus 네임스페이스에 대해 처리량 단위를 20 개까지 구입할 수 있으며 Azure 계정의 처리량 단위 제한은 20 개입니다. 이러한 처리량 단위는 지정된 네임스페이스의 모든 이벤트 허브에서 공유됩니다.

처리량 단위는 최상의 노력 방식으로 프로비전되며 항상 즉시 구입할 수 있는 것은 아닙니다. 특정 용량이 필요한 경우 해당 처리량 단위를 미리 구입하는 것이 좋습니다. 처리량 단위가 20 개보다 많이 필요한 경우에는 Microsoft Azure 서비스 버스 지원에 문의하여 처음 100 개 처리량 단위까지 약정을 통해 처리량 단위를 20 개 단위로 더 구입할 수 있습니다. 처리량 단위가 100 개보다 더 많이 필요한 경우에는 100 개 단위로 더 구입할 수도 있습니다.

이벤트 허브를 가장 효율적으로 확장할 수 있도록 처리량 단위와 파티션 간의 균형을 적절하게 조정하는 것이 좋습니다. 파티션 하나의 최대 확장 단위는 처리량 단위 하나입니다. 처리량 단위의 수는 이벤트 허브의 파티션 수 이하여야 합니다.

자세한 가격 정보는 이벤트 허브 가격을 참조하세요.

### 게시자 정책

이벤트 허브에서는 게시자 정책을 통해 이벤트 생산자를 세부적으로 제어할 수 있습니다. 게시자 정책은 다수의 독립 이벤트 생산자가 원활하게 작동할 수 있도록 하는 런타임 기능 집합입니다. 게시자 정책을 사용하는 경우 각 게시자는 다음 메커니즘을 사용하여 이벤트 허브에 이벤트를 게시할 때 고유한 자체 식별자를 사용합니다.

복사

```
//<my namespace>.servicebus.windows.net/<event hub name>/publishers/<my publisher name>
```

게시자 이름을 미리 만들 필요는 없지만 게시자 ID 의 독립성을 보장하려면 이벤트를 게시할 때 사용하는 SAS 토큰과 게시자 이름이 일치해야 합니다. SAS 에 대한 자세한 내용은 Service Bus 의 공유 액세스 서명 인증을 참조하세요. 게시자 정책을 사용할 때 **PartitionKey** 값은 게시자 이름으로 설정됩니다. 이러한 값이 일치해야 정책이 정상적으로 작동합니다.

## 요약

Azure 이벤트 허브는 규모에 관계없이 일반적인 응용 프로그램 및 사용자 워크플로 모니터링에 사용할 수 있는 대규모 이벤트 및 원격 분석 수집 서비스를 제공합니다. 짧은 대기 시간에 대규모로 게시/구독하는 기능을 제공하는 이벤트 허브는 빅 데이터의 "진입점" 역할을 합니다. 게시자 기반 ID 및 해지 목록을 통해 이러한 기능을 일반적인 사물 인터넷 시나리오로 확장할 수 있습니다. 이벤트 허브 응용 프로그램을 개발하는 방법에 대한 자세한 내용은 이벤트 허브 프로그래밍 가이드를 참조하세요.