

한국 마이크로소프트

Microsoft Technical Trainer

Enterprise Skills Initiative

Microsoft Azure

Azure Queue Storage

이 문서는 Microsoft Technical Trainer팀에서 ESI 교육 참석자분들에게 제공해 드리는 문서입니다.

요약

이 내용들은 표시된 날짜에 Microsoft에서 검토된 내용을 바탕으로 하고 있습니다. 따라서, 표기된 날짜 이후에 시장의 요구사항에 따라 달라질 수 있습니다. 이 문서는 고객에 대한 표기된 날짜 이후에 변화가 없다는 것을 보증하지 않습니다.

이 문서는 정보 제공을 목적으로 하며 어떠한 보증을 하지는 않습니다.

저작권에 관련된 법률을 준수하는 것은 고객의 역할이며, 이 문서를 마이크로소프트의 사전 동의 없이 어떤 형태(전자 문서, 물리적인 형태 막론하고) 어떠한 목적으로 재 생산, 저장 및 다시 전달하는 것은 허용되지 않습니다.

마이크로소프트는 이 문서에 들어있는 특허권, 상표, 저작권, 지적 재산권을 가집니다. 문서를 통해 명시적으로 허가된 경우가 아니면, 어떠한 경우에도 특허권, 상표, 저작권 및 지적 재산권은 다른 사용자에게 허여되지 않습니다.

© 2022 Microsoft Corporation All right reserved.

Microsoft®는 미합중국 및 여러 나라에 등록된 상표입니다.

이 문서에 기재된 실제 회사 이름 및 제품 이름은 각 소유자의 상표일 수 있습니다.

문서 작성 연혁

날짜	버전	작성자	변경 내용
2022.02.06	0.6.0	우진환	TASK 01, TASK 02 작성
2022.02.07	1.0.0	우진환	TASK 03 작성
2022.04.05	1.1.0	우진환	리소스 그룹 이름 변경 및 Azure 포털 UI 변경 적용

목차

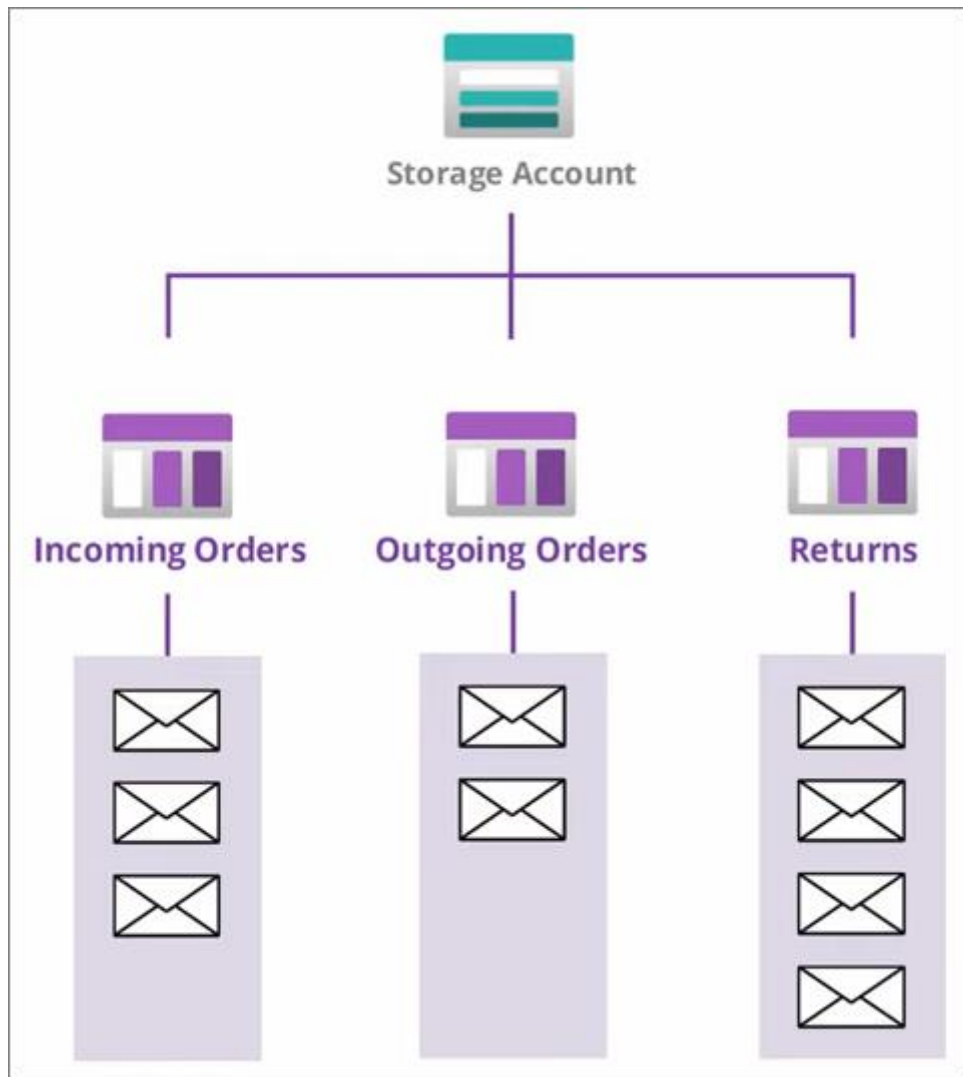
작동 방법.....	5
QUEUE를 사용하는 이유.....	7
재시도 패턴(RETRY PATTERN)	8
TASK 01. AZURE STORAGE QUEUE 만들기	9
TASK 02. QUEUE 데이터 관리	10
TASK 03. FAN-OUT 시나리오 구성	13
TASK 04. 리소스 정리.....	34

Azure Queue Storage는 많은 수의 메시지를 저장할 목적으로 만들어진 서비스이며 Azure 스토리지 계정의 하위 서비스입니다. Azure Queue Storage는 HTTP 또는 HTTPS를 통한 인증된 호출을 사용하여 전세계 어디에서도 메시지에 액세스할 수 있습니다. 큐(queue) 메시지의 최대 크기는 64KB입니다. 큐(queue)는 스토리지 계정의 총 용량 제한까지 수백만 개의 메시지가 포함될 수 있습니다. 큐(queue)는 일반적으로 비동기식으로 처리할 작업의 백로그(backlog)를 만드는데 사용됩니다.

작동 방법

큐(queue)는 스토리지 계정의 일부이므로 애플리케이션 보안, 높은 확장성과 같은 스토리지 계정의 모든 기능을 사용하지만 한 가지 중요한 점은 스토리지 계정 당 초당 처리할 수 있는 기본 최대 요청률은 20,000개 요청입니다.

- 각 스토리지 계정 아래에 하나 이상의 큐(queue)를 만들 수 있습니다. 일반적으로 들어오는 주문, 나가는 주문, 반품 및 처리와 같은 비즈니스 로직에 대해 별도의 큐(queue)를 사용하며, 각 큐(queue)는 메시지의 컨테이너입니다.
- Azure Queue Storage는 최대 500TiB 데이터를 저장할 수 있지만 단일 큐(queue)는 초당 2,000개의 메시지(1KiB)만 처리할 수 있으므로 더 많은 메시지를 처리해야 하는 경우 다른 접근 방법을 고려해야 합니다
- 각 큐(queue)에는 메시지(message)가 있으며 메시지는 JSON 바이너리, 텍스트 CSV 등 모든 종류의 데이터입니다.
- 단일 메시지(message)는 최대 64KB이며 이 이상의 크기를 초과할 수 없습니다. 이는 큐를 통해서는 핵심 정보만 전송하고 나머지 정보는 데이터베이스나 다른 데이터 저장 솔루션에서 가져와야 하기 때문입니다.
- 큐(queue)에는 TTL (Time-to-Live)이 있으며 메시지를 수신하지 않으면 기본적으로 큐에서 무기한으로 유지됩니다. api-version 2017-07-27 이후부터는 무기한 유지되며 이전 버전에서는 기본 7일간 유지됩니다.



Azure는 스토리지 큐(Storage Queue)와 서비스 버스 큐(Service Bus Queue)의 두 가지 유형의 큐 메커니즘을 지원합니다.

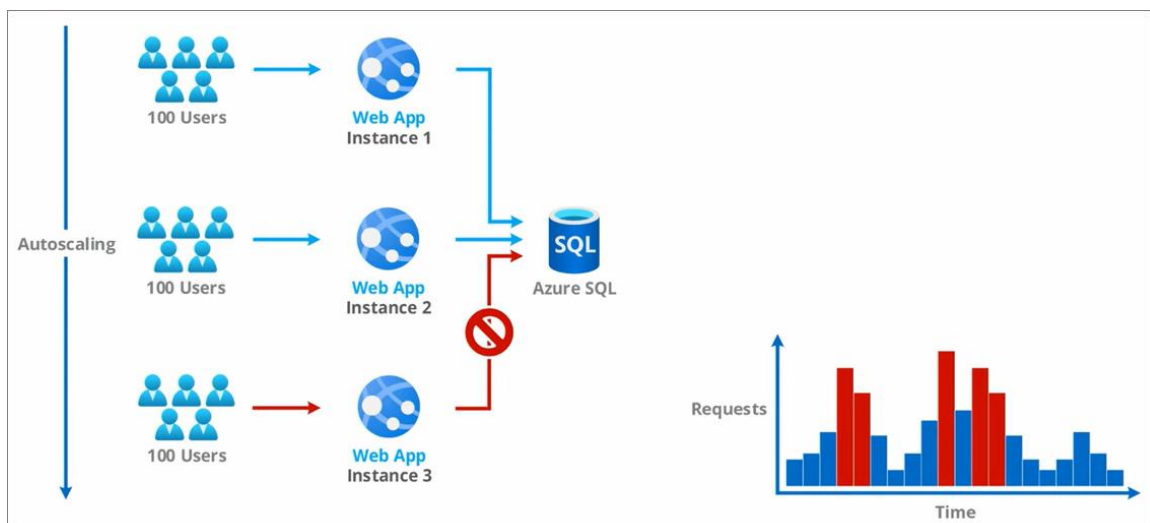
- Storage Queue는 Azure 스토리지 인프라의 일부이며 많은 수의 메시지를 저장할 수 있습니다. HTTP/HTTPS를 사용하여 인증된 호출을 통해 전 세계 어디에서나 메시지에 액세스할 수 있으며 메시지의 최대 크기는 64KB입니다. Queue에는 스토리지 계정의 총 용량 제한까지 수백만 개의 메시지가 포함될 수 있습니다. Queue는 일반적으로 비동기식으로 처리할 작업의 백로그(backlog)를 만드는데 사용됩니다.
- Service Bus Queue는 queuing, publish/subscribe 및 고급 통합 패턴을 지원하는 광범위한 Azure 메시징 인프라의 일부입니다. Service Bus Queue는 여러 커뮤니케이션 프로토콜, 데이터 계약, 신뢰 도메인, 네트워크 환경에 걸쳐 있는 애플리케이션 및 애플리케이션의 구성 요소를 통합하도록 디자인되었습니다.

서비스	사용 시나리오
Storage Queue	<ul style="list-style-type: none"> 애플리케이션이 Queue에 80GB 이상의 메시지를 저장해야 하는 경우 애플리케이션에서 Queue의 메시지 처리 진행률을 추적하려는 경우. 메시지를 처리하는 워커(worker)가 충돌하는 경우 유용함. 다른 워커는 해당 정보를 사용하여 이전 워커가 중단한 부분부터 작업을 계속할 수 있음

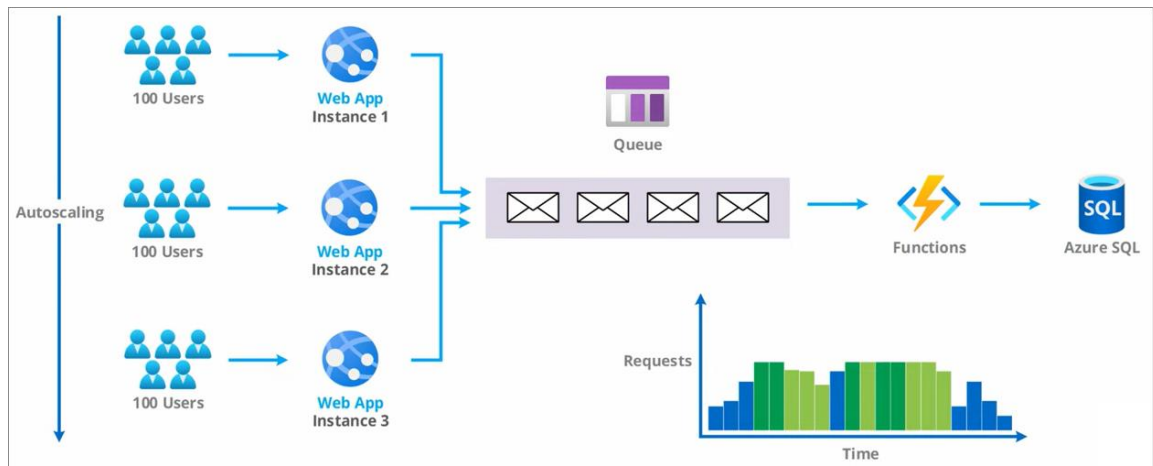
	<ul style="list-style-type: none"> Queue 에서 실행된 모든 트랜잭션의 서버측 로그가 필요한 경우
Service Bus Queue	<ul style="list-style-type: none"> Queue 를 폴링(polling)하지 않고 메시지를 수신해야 하는 경우. Service Bus 는 Service Bus 에서 지원되는 TCP 기반 프로토콜을 사용하여 긴 폴링 수신 작업을 사용할 수 있음 솔루션에서 FIFO 순서가 보장되는 Queue 가 필요한 경우 솔루션에서 자동 중복 감지를 지원해야 하는 경우 애플리케이션이 메시지를 병렬 장기 실행 스트림으로 처리하기 원하는 경우 Queue 에서 여러 메시지를 보내거나 받을 때 솔루션에 트랜잭션 동작과 원자성(atomicity)이 필요한 경우 애플리케이션이 64KB 를 초과하지만 256KB 이하인 메시지를 처리하는 경우 Queue 에 역할 기반 액세스 모델을 제공하고 발신자와 수신자에 대해 서로 다른 권한을 제공해야 하는 경우 Queue 크기가 80GB 이상 커지지 않고 AMQP 1.0 표준 기반 메시징 프로토콜을 사용하고자 하는 경우

Queue를 사용하는 이유

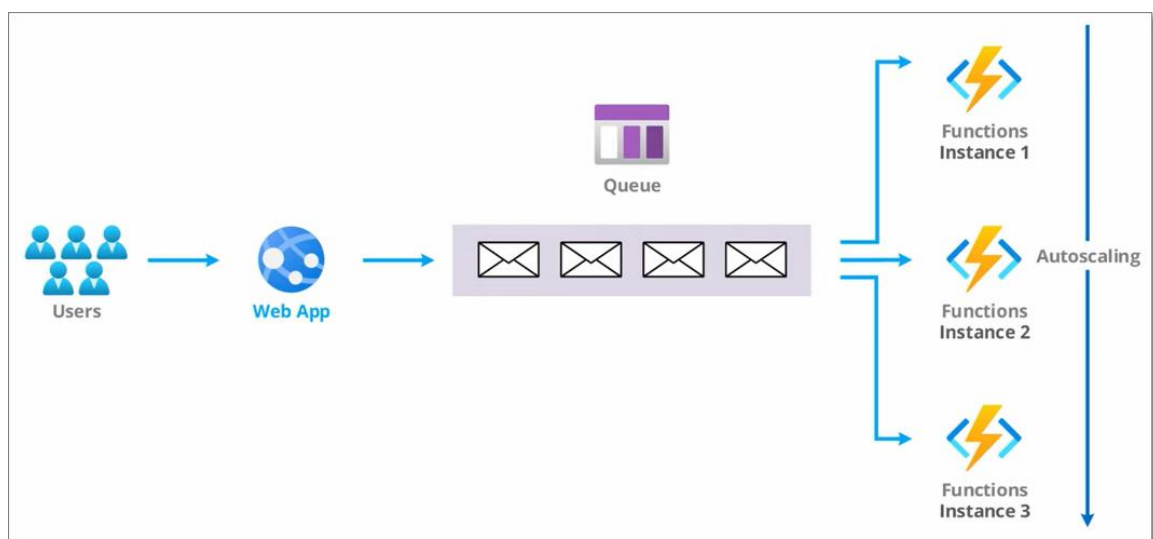
예를 들어 아래와 같이 SQL에 데이터를 저장하고 각 100명의 사용자에게 대해 별도의 웹 애플리케이션 인스턴스를 가지는 시나리오를 생각해 보겠습니다. 이 시나리오에서 사용자가 100명을 초과할 때마다 웹 애플리케이션은 자동으로 확장됩니다. 하지만 트래픽이 많아지는 특정 기간 동안 여러 가지 문제가 발생할 수 있습니다. SQL 데이터베이스가 모든 사용자의 요청을 처리할 수 있을 만큼 충분한 크기를 제공하지 못하거나 백 엔드 API가 충분하지 않을 수 있습니다. 이 경우 웹 애플리케이션이 아무리 많더라도 사용자 요청은 실패하게 됩니다.



위와 같은 시나리오에서 데이터베이스를 증가하지 않고 문제를 해결할 수 있는 방법이 있습니다. 웹 애플리케이션 뒤에 간단히 Queue를 추가하고 모든 메시지를 Queue로 출력한 다음, 데이터베이스가 처리할 수 있는 속도로 이 Queue에서 메시지를 가져오는 Function Handler나 API를 배치할 수 있습니다. 이를 통해 애플리케이션 워크로드의 부하 수준을 평평하게 만들 수 있습니다. 물론 이 경우 사용자에게 대한 요청이 기존 구성보다 느려질 수 있지만 큰 비용을 들이지 않고 기존의 성능 이슈를 해결할 수 있게 됩니다. 이러한 패턴을 큐 기반 부하 평준화 패턴(Queue-Based Load Leveling pattern)이라고 합니다.



또 다른 Queue 사용 시나리오로 fan-out 시나리오를 고려할 수 있습니다. 아래와 같이 사용자는 웹 애플리케이션에 요청을 보내고 웹 애플리케이션은 이 요청 메시지를 Queue에 넣습니다. 그런 다음 Queue의 크기에 따라 확장되는 Function을 만들고 메시지를 개별적으로 선택한 다음 병렬로 처리하도록 구성할 수 있습니다. 이를 경쟁 소비자 패턴(Competing Consumer pattern)이라고 합니다. 이는 애플리케이션을 개별적으로 확장하는 방법이며 Azure에서 serverless 가 실제로 작동하는 방식의 핵심 기능 중 하나입니다.



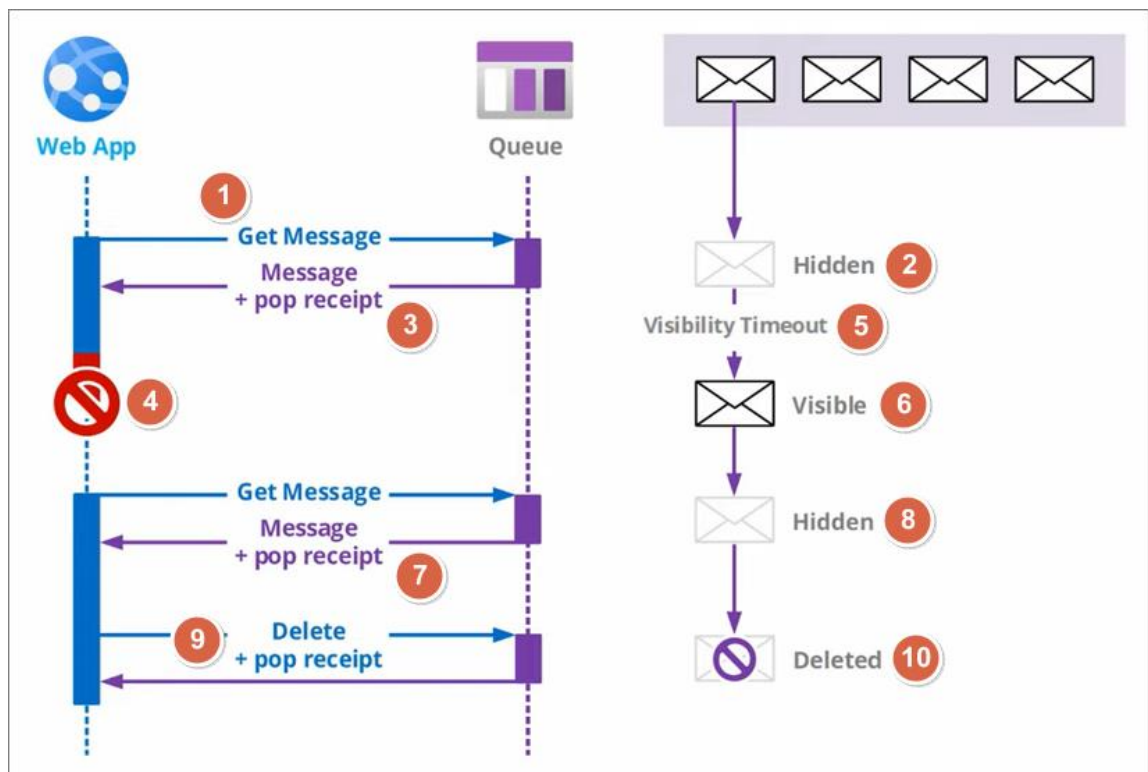
재시도 패턴(Retry Pattern)

Queue와 웹 애플리케이션이 있고 웹 애플리케이션이 Queue에서 메시지를 가져오도록 요청을 보내는 경우 다음과 같은 프로세스가 진행됩니다.

- 웹 애플리케이션이 Queue에 메시지를 가져오도록 요청합니다.
- Queue 서비스는 메시지를 숨김(hidden) 상태로 넣고 ③클라이언트에게 메시지와 함께 **pop receipt**를 반환합니다. **pop receipt** 기능은 개발자가 추가 처리를 위해 큐에 있는 메시지(enqueued message)를 쉽게 식별할 수 있도록 해주는 도구입니다. 따라서 현재 메시지를 수신한 클라이언트가 메시지를 처리 중일 때에는 다른 클라이언트 애플리케이션에서 이 메시지를 볼 수 없으며 숨겨져 있습니다. 하지만 이 메시지는

숨겨져 있는 것이지 삭제되는 것이 아닙니다.

- 메시지를 수신한 애플리케이션이 메시지를 처리하지 못하고 문제가 발생하는 경우, ⑤Queue는 **visibility timeout**을 사용하여 문제를 해결합니다.
- ⑥따라서 특정 시간이 지나면 메시지가 Queue에 다시 표시됩니다.
- ⑦웹 애플리케이션이나 이 Queue에 연결된 다른 애플리케이션은 해당 메시지와 **pop receipt**를 Queue에서 다시 가져오고 ⑧메시지는 숨김 상태가 됩니다. 그런 다음 애플리케이션은 메시지 처리를 다시 진행합니다.
- ⑨애플리케이션에서 메시지가 제대로 처리되면 이번에 받은 **pop receipt**와 함께 Queue에 삭제 요청을 보낼 수 있으며 ⑩그 다음에 메시지가 삭제됩니다.



이 프로세스에서 중요한 것은 성공적으로 메시지를 처리했다는 메시지를 받은 후 실제로 다시 응답해야 한다는 것입니다. 메시지가 정상적으로 처리될 때까지 Queue의 메시지는 숨김 상태로 있기 때문에 매우 탄력적인 아키텍처를 제공하여 애플리케이션이 여러 번 메시지 처리를 재시도할 수 있습니다.

TASK 01. Azure Storage Queue 만들기

이 작업에서는 Queue 스토리지를 만듭니다. Azure 스토리지 계정에 Queue가 포함되어 있기 때문에 실제로는 Azure 스토리지 계정을 만듭니다.

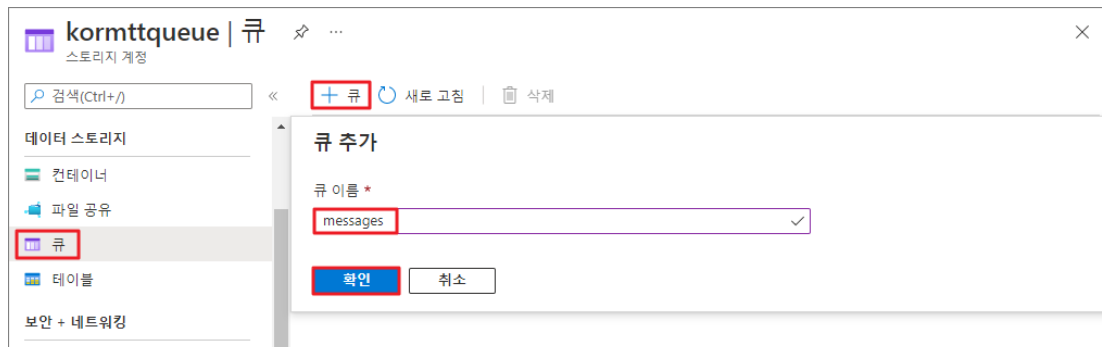
1. Azure 포털에서 [리소스 만들기]를 클릭한 후 "스토리지 계정"을 검색하고 클릭합니다. [스토리지 계정] 블레이드에서 [만들기]를 클릭합니다.

2. [저장소 계정 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를 클릭합니다.
 - [프로젝트 정보 - 리소스 그룹]: "새로 만들기"를 클릭한 후 "03_storageQueueRg"를 입력합니다.
 - [인스턴스 정보 - 스토리지 계정 이름]: 중복되지 않는 고유한 이름을 입력합니다.
 - [인스턴스 정보 - 지역]: (Asia Pacific) Korea Central
 - [인스턴스 정보 - 성능]: 표준
 - [인스턴스 정보 - 중복]: LRS(로컬 중복 스토리지)

TASK 02. Queue 데이터 관리

이 작업에서는 Storage Queue에 새 큐를 만들고 메시지를 생성하는 작업을 진행합니다.

1. 앞서 만든 [스토리지 계정] 블레이드로 이동합니다. [데이터 스토리지 - 큐]로 이동한 후 메뉴에서 [큐]를 클릭합니다. [큐 추가]에서 "messages"를 입력한 후 [확인]을 클릭합니다.

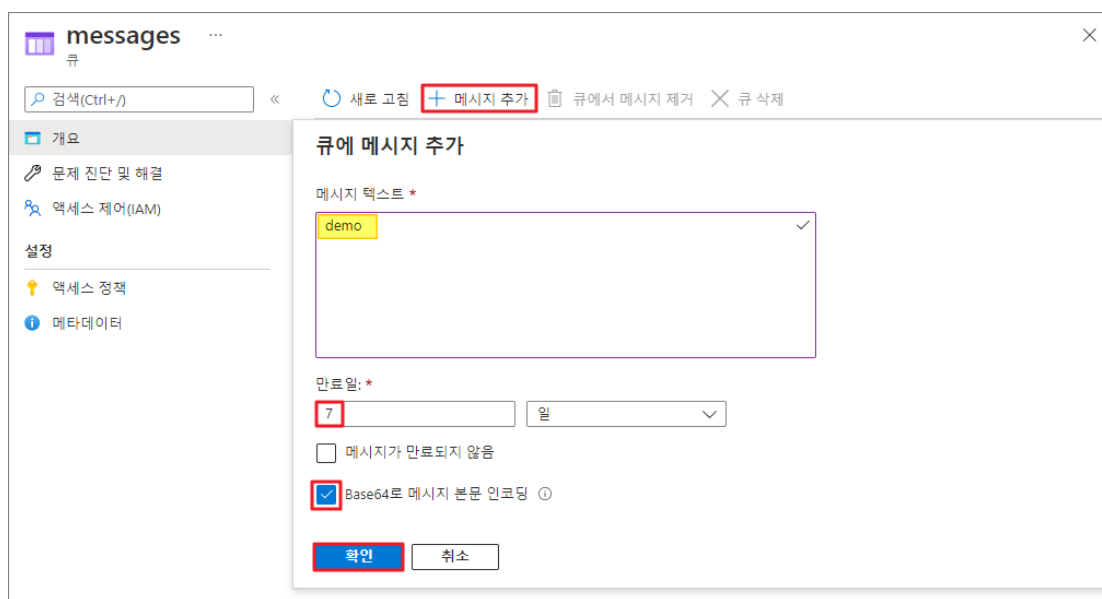


2. 새로 만든 큐에 새 URL이 생성된 것을 확인하고 이 큐를 클릭합니다.

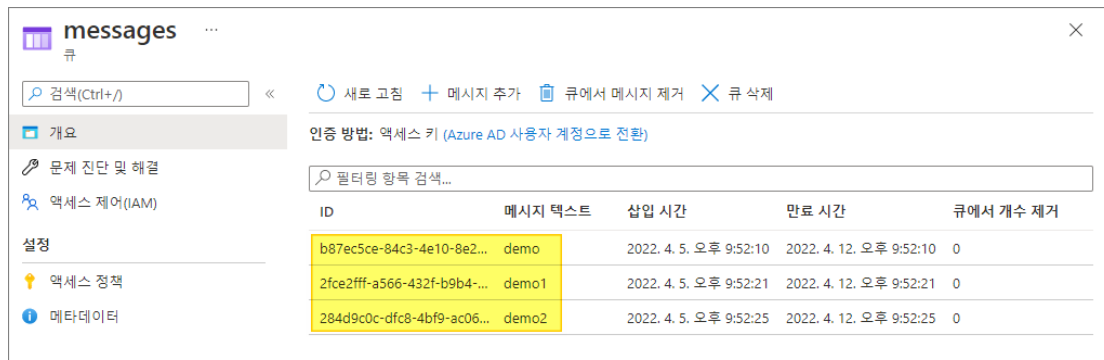


3. [messages] 큐 블레이드의 메뉴에서 [메시지 추가]를 클릭합니다. [큐에 메시지 추가]에서 아래와 같은 내용을 입력하고 [확인]을 클릭합니다.

- 메시지 텍스트: demo
- 만료일: 기본 설정으로 7일 후에 메시지가 만료됩니다.
- 메시지가 만료되지 않음: 이 옵션을 체크하면 큐의 메시지를 삭제하지 않는 이상 메시지는 삭제되지 않습니다.
- Base64로 메시지 본문 인코딩: 메시지 본문을 인코딩할 것인지 설정합니다.



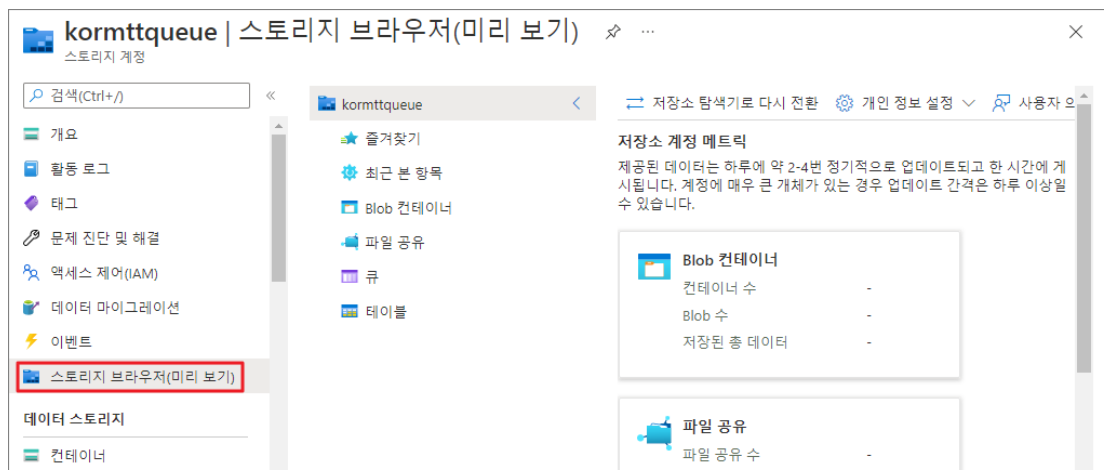
4. 동일한 작업을 반복하여 메시지를 몇 개 더 추가합니다.



5. 큐에 추가된 특정 메시지를 삭제하고자 하는 경우 해당 메시지를 선택하고 메뉴에서 [큐에서 메시지 제거(Dequeue message)]를 클릭합니다. 특정 메시지를 삭제한 후 큐의 전체 메시지를 삭제하고자 하는 경우 [큐 삭제]를 클릭하여 전체 메시지를 삭제할 수 있습니다.



6. [스토리지 계정] 블레이드로 돌아간 후 [스토리지 브라우저]를 클릭합니다.



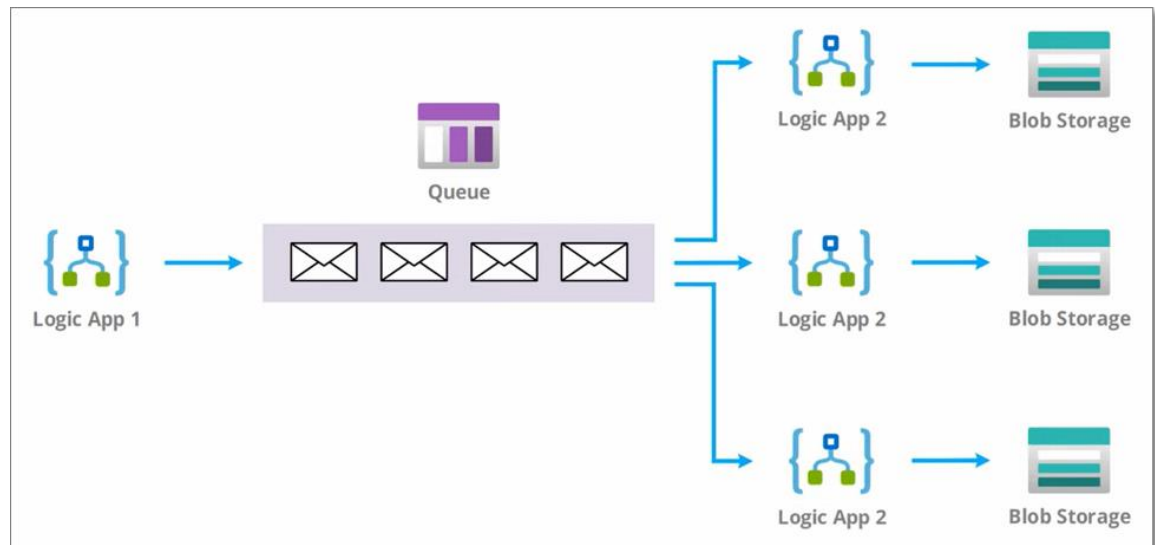
7. [스토리지 브라우저]에서 [큐 - messages]로 이동합니다. Azure 포털과 동일한 메뉴를 확인할 수 있습니다. [메시지 추가]를 클릭한 후 메시지 텍스트에 "demo"를 입력하고 [확인]을 클릭합니다.



TASK 03. Fan-out 시나리오 구성

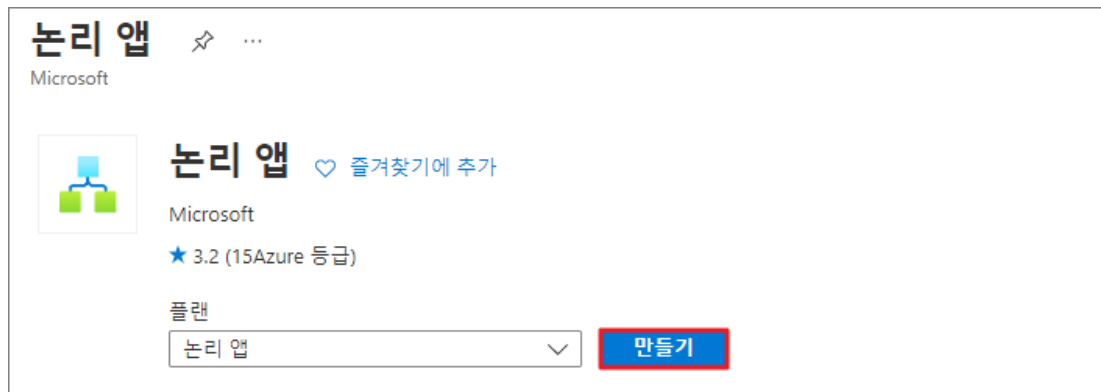
이 작업에서는 아래와 같은 시나리오를 구성합니다.

- 첫 번째 Logic App에서 Storage Queue에 메시지를 전송합니다.
- 두 번째 Logic App에서 Storage Queue의 메시지를 가져올 수 있도록 구성합니다.
- 두 번째 Logic App을 수정하여 Storage Queue에서 가져온 메시지를 Storage Blob에 파일로 기록하는 작업을 추가합니다.



이 작업을 위해 두 개의 Logic App이 필요합니다. 다음과 같은 작업을 진행합니다.

1. Azure 포털에서 [리소스 만들기]를 클릭한 후 "논리 앱"을 검색합니다. [논리 앱] 블레이드에서 [만들기]를 클릭합니다.



2. [논리 앱 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를 클릭합니다.

- [프로젝트 세부 정보 - 리소스 그룹]: 06_storageQueueRg
- [인스턴스 정보 - 논리 앱 이름]: sendQueue
- [인스턴스 정보 - 지역]: Korea Central
- [인스턴스 정보 - Log analytics 사용 설정]: 아니요
- [플랜 - 플랜 유형]: 소비

논리 앱 만들기 ...

기본 태그 검토 + 만들기

손쉬운 리소스 관리, 배포 및 공유를 위해 논리적 단위로 워크플로를 그룹화할 수 있는 논리 앱을 만듭니다. 워크플로를 사용하면 업무상 중요한 앱 및 서비스를 Azure Logic Apps에 연결하고 코드를 한 줄도 작성하지 않고 워크플로를 자동화할 수 있습니다.

프로젝트 세부 정보

배포된 리소스와 비용을 관리할 구독을 선택합니다. 폴더 같은 리소스 그룹을 사용하여 모든 리소스를 정리 및 관리합니다.

구독 * ⓘ Azure Pass - 스폰서십

리소스 그룹 * ⓘ 06_storageQueueRg
[새로 만들기](#)

인스턴스 정보

논리 앱 이름 * sendQueue

지역 * Korea Central

Log analytics 사용 설정 * ☐ 예 ☒ 아니요

플랜

선택한 플랜 유형에 따라 앱 확장 방식, 사용하도록 설정된 기능 및 가격 책정 방식이 결정됩니다.

플랜 유형 * ☒ 소비: 입문 수준에 가장 적합합니다. 워크플로가 실행되는 만큼만 비용을 지불하세요.
☐ 기준: 이벤트 기반 확장 및 네트워킹 격리를 통해 엔터프라이즈 수준의 서버리스 애플리케이션에 가장 적합합니다.

[클래식 소비 경험 만들기를 찾고 계십니까? 여기를 클릭](#)

3. 동일한 방법으로 두 번째 논리 앱을 만듭니다. [논리 앱 만들기] 블레이드의 [기본] 탭에서 아래와 같이

구성한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를 클릭합니다.

- [프로젝트 세부 정보 - 리소스 그룹]: 06_storageQueueRg
- [인스턴스 정보 - 논리 앱 이름]: receiveQueue
- [인스턴스 정보 - 지역]: Korea Central
- [인스턴스 정보 - Log analytics 사용 설정]: 아니요
- [플랜 - 플랜 유형]: 소비

논리 앱 만들기 ...

기본 태그 검토 + 만들기

순쉬운 리소스 관리, 배포 및 공유를 위해 논리적 단위로 워크플로를 그룹화할 수 있는 논리 앱을 만듭니다. 워크플로를 사용하면 업무상 중요한 앱 및 서비스를 Azure Logic Apps에 연결하고 코드를 한 줄도 작성하지 않고 워크플로를 자동화할 수 있습니다.

프로젝트 세부 정보

배포된 리소스와 비용을 관리할 구독을 선택합니다. 풀더 같은 리소스 그룹을 사용하여 모든 리소스를 정리 및 관리합니다.

구독 * ① Azure Pass - 스폰서십

리소스 그룹 * ① 06_storageQueueRg
[새로 만들기](#)

인스턴스 정보

논리 앱 이름 * receiveQueue

지역 * Korea Central

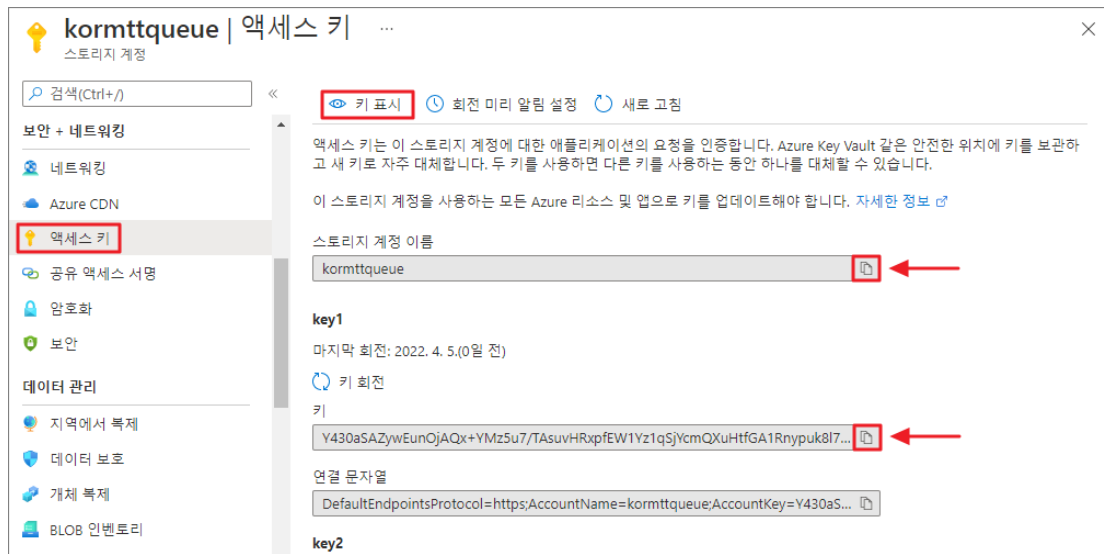
Log analytics 사용 설정 * ☐ 예 ☒ 아니요

플랜

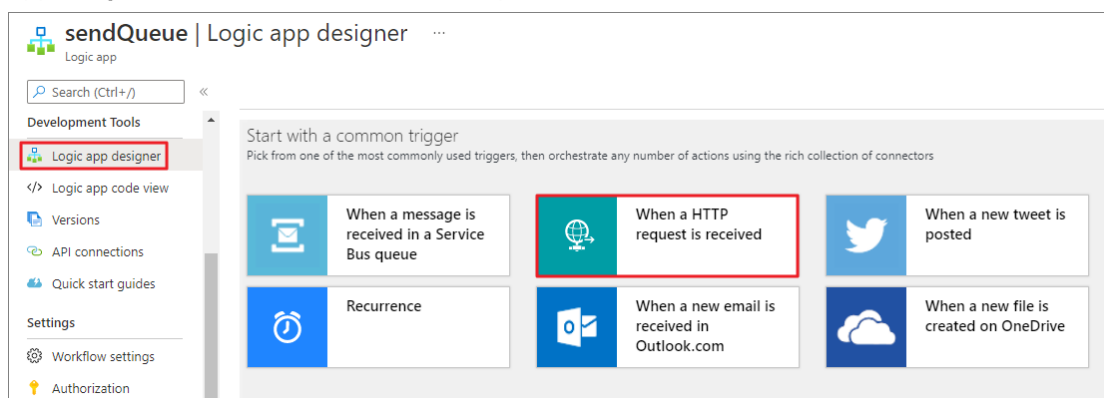
선택한 플랜 유형에 따라 앱 확장 방식, 사용하도록 설정된 기능 및 가격 책정 방식이 결정됩니다.

플랜 유형 * ☒ 소비: 입문 수준에 가장 적합합니다. 워크플로가 실행되는 만큼만 비용을 지불하세요.
☐ 기준: 이벤트 기반 확장 및 네트워킹 격리를 통해 엔터프라이즈 수준의 서버리스 애플리케이션에 가장 적합합니다.
☐ 클래식 소비 경험 만들기를 찾고 계십니까? [여기를 클릭](#)

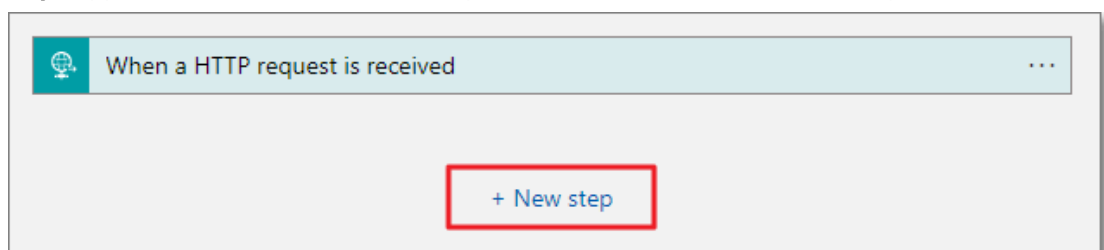
4. [스토리지 계정] 블레이드의 [보안 + 네트워킹 - 액세스 키]로 이동합니다. 메뉴에서 [키 표시]를 클릭한 후 스토리지 계정 이름, Key1의 키 값을 메모장에 복사합니다.



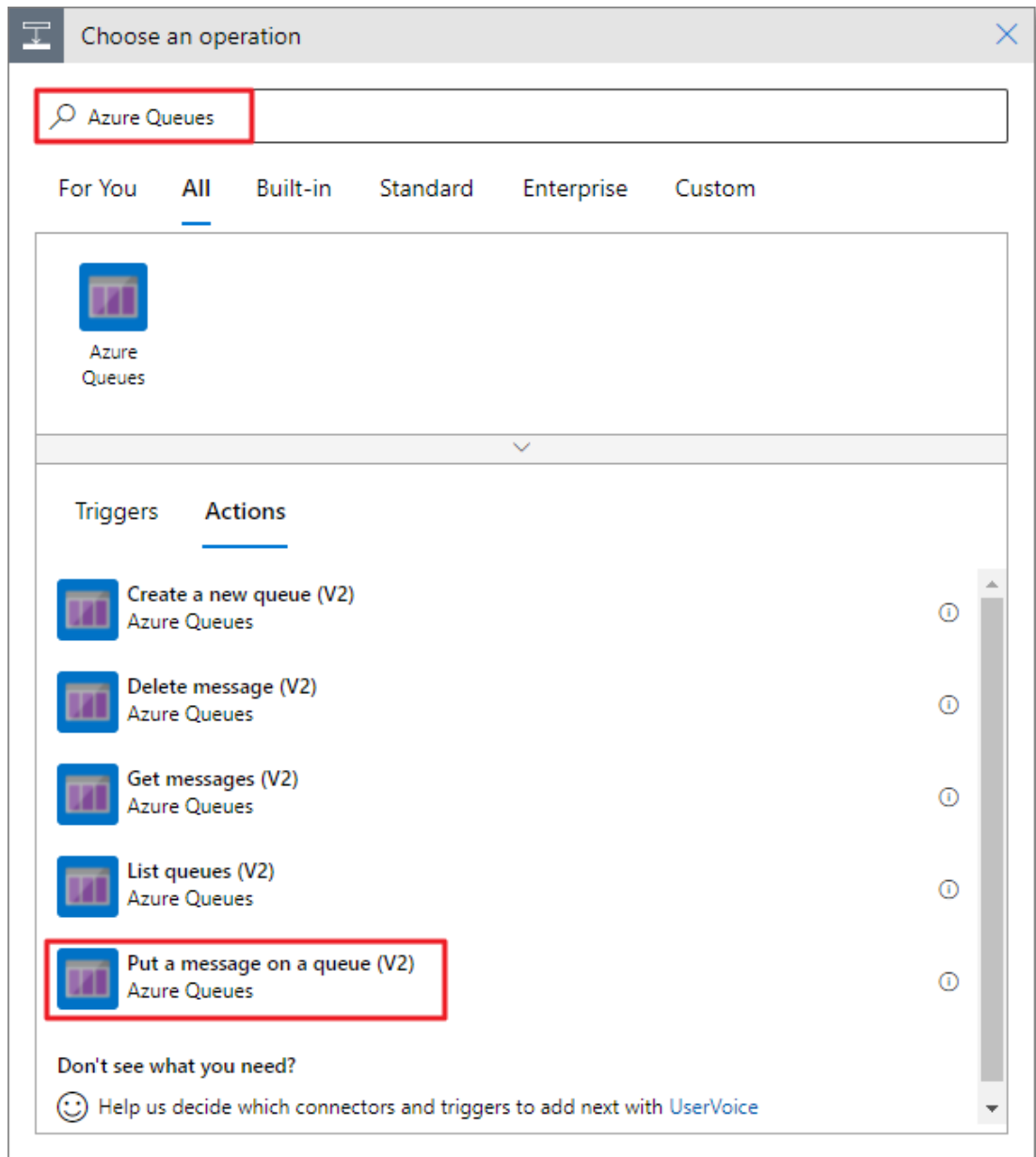
5. Logic App 작업을 할 때는 Azure 포털 페이지를 영어로 변경하는 것이 좋습니다. 포털의 언어를 영어로 변경한 후 처음 만들었던 [sendQueue Logic app] 블레이드로 이동합니다. [Development Tools - Logic app designer]로 이동한 후 [When a HTTP request is received] 타일을 클릭합니다.



6. Logic App 디자이너에서 [When a HTTP request is received] 타일 아래의 [New step]을 클릭합니다.



7. [Choose an operation] 타일의 검색창에 "Azure Queues"를 검색한 후 [Actions] 탭에서 [Put a message on an queue (v2)]를 클릭합니다.



8. [Azure Queues] 타일에서 아래와 같이 구성한 후 [Create]를 클릭합니다.

- Connection name: queueConnection
- Authentication Type: Access Key
- Azure Storage Account name: 앞서 [스토리지 계정] 블레이드에서 복사한 스토리지 계정 이름을 붙여 넣습니다.
- Azure Storage Account Access Key: [스토리지 계정] 블레이드에서 복사한 **key1**의 값을 붙여 넣습니다.

Azure Queues

* Connection name: queueConnection

* Authentication Type: Access Key

* Azure Storage Account name: kormttqueue

Azure Storage Account Access Key:

Create

9. [Put a message on a queue (V2)] 타일에서 아래와 같이 구성합니다.

- Storage account name: 드롭다운 메뉴를 클릭한 후 앞서 만들었던 스토리지 계정을 선택합니다.
- Queue Name: messages
- Message: "demo"를 입력한 후 [Add dynamic content]를 클릭합니다. [Expression] 탭에서 "rand" 함수를 검색하고 `rand(1, 10000)` 값을 입력한 후 [OK]를 클릭합니다.

Put a message on a queue (V2)

* Storage account name: Use connection settings (kormttqueue)

* Queue Name: messages

* Message: demo

Connected to queueConnection. Change connection.

+ New step

Dynamic content: Expression

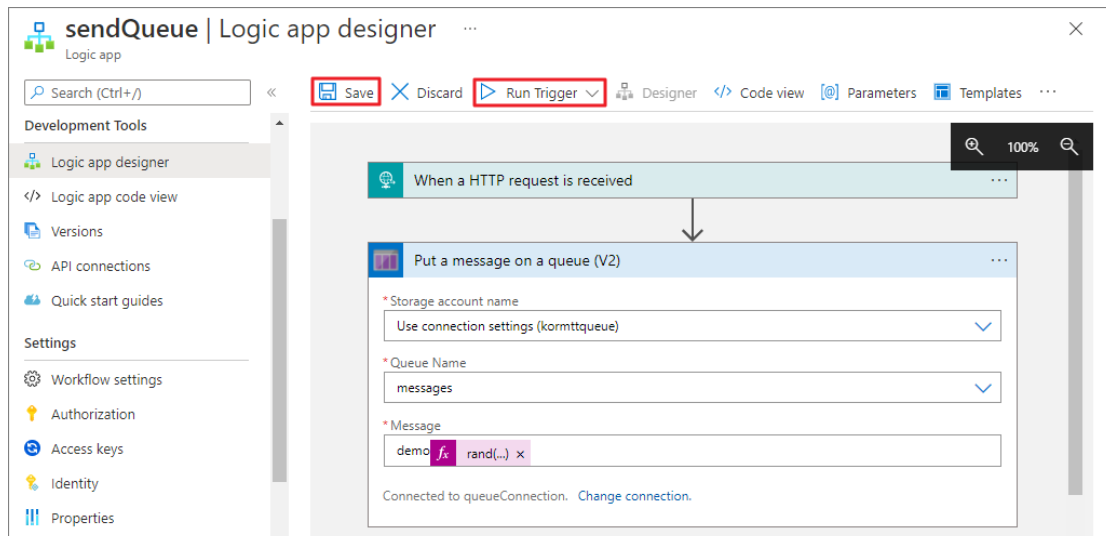
fx rand(1, 10000)

OK

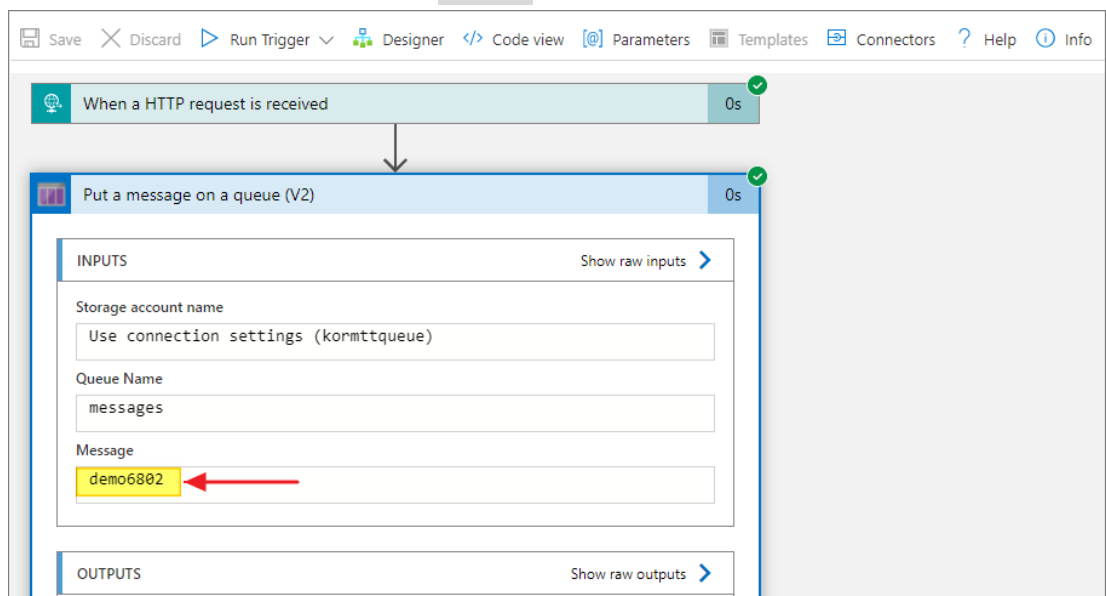
String functions: concat(text_1, text_2?, ...)

Collection: contains(collection, value)

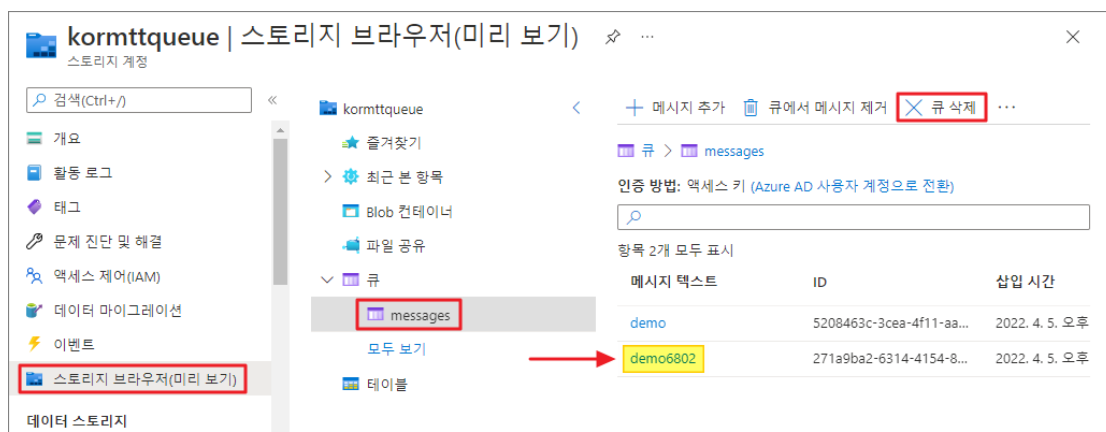
10. Logic App 디자이너 메뉴에서 [Save]를 클릭한 후 [Run Trigger - Run]을 클릭합니다.



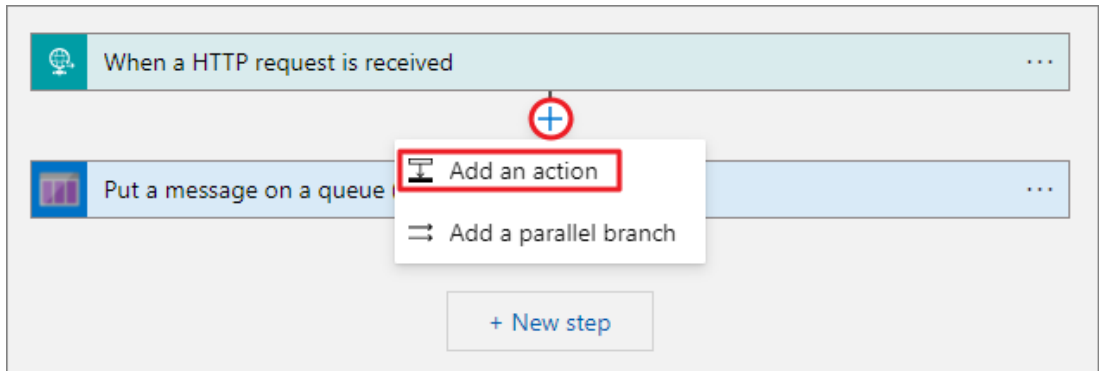
11. Logic App 실행 결과에서 임의로 생성된 Message 값을 확인합니다.



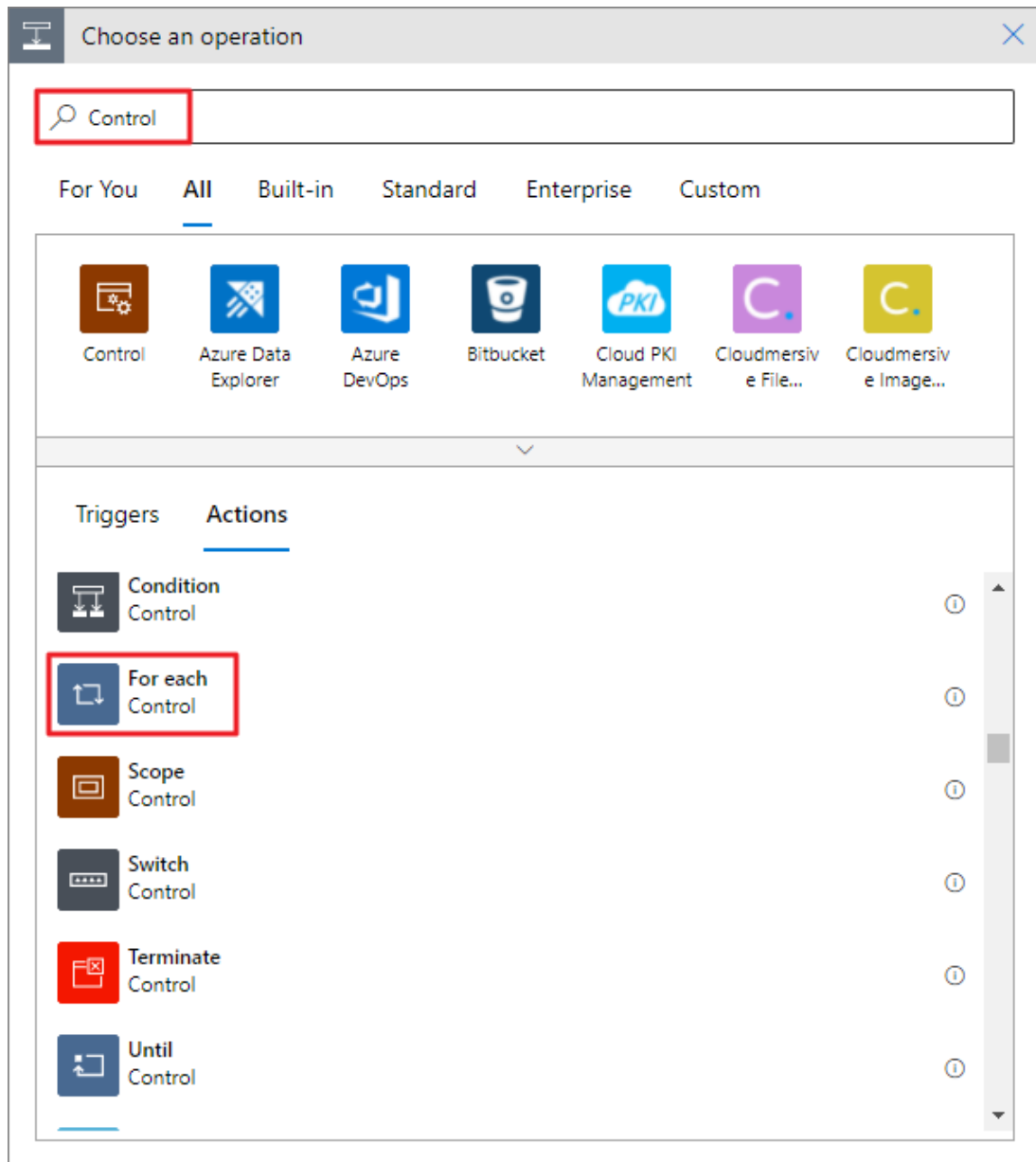
12. [스토리지 계정] 블레이드로 이동한 후 [스토리지 브라우저]를 클릭합니다. [큐 - messages]로 이동한 후 Logic App을 통해 생성된 메시지가 표시되는 것을 확인합니다. [큐 삭제]를 클릭하여 큐의 모든 메시지를 삭제합니다.



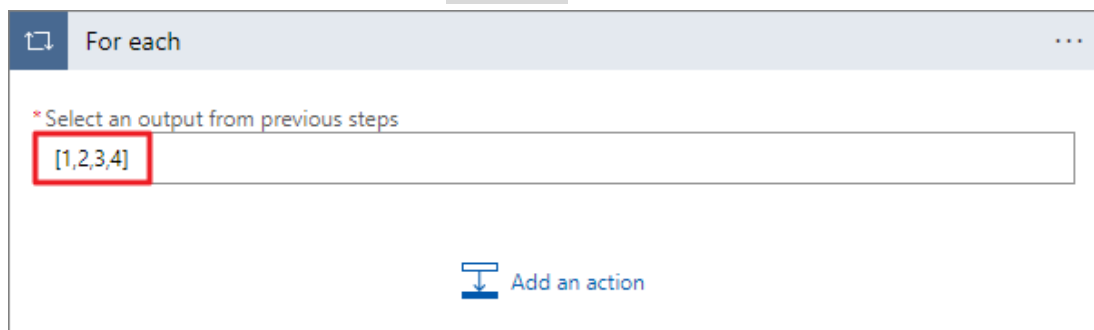
13. [sendQueue Logic app] 블레이드의 [Development Tools - Logic app designer]로 이동합니다. Logic App 디자이너에서 [When a HTTP request is received] 타일 아래의 [+ - Add an action] 버튼을 클릭합니다.



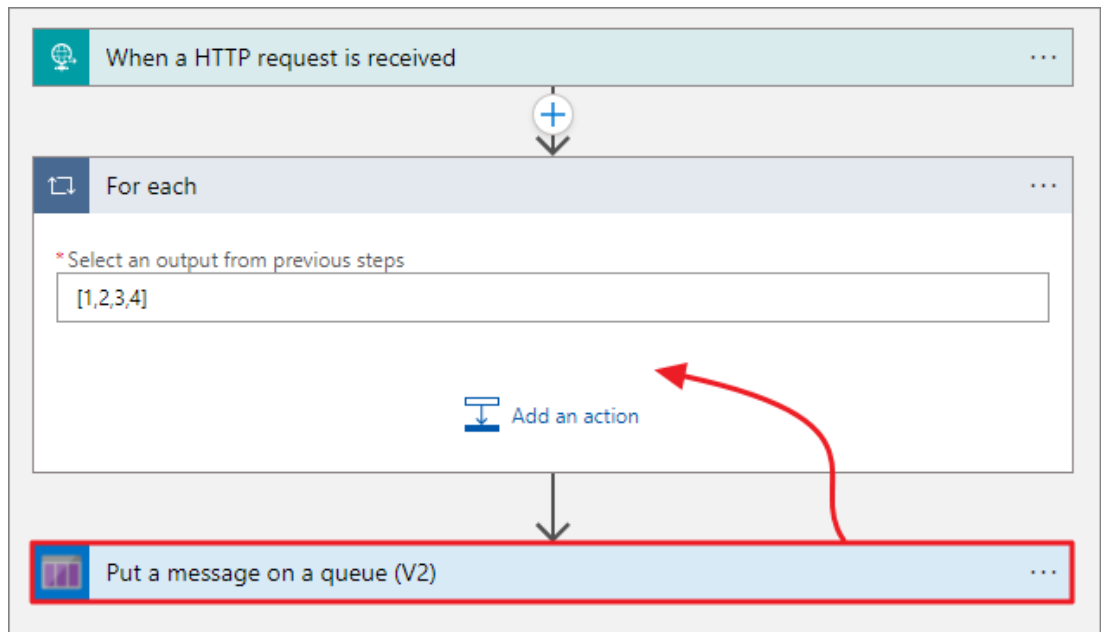
14. Storage Queue에 반복적으로 메시지를 전송하기 위해 반복문을 추가할 것입니다. [Choose an operation] 타일의 검색창에 "Control"을 검색하고 [Actions] 탭에서 [For each]를 클릭합니다.



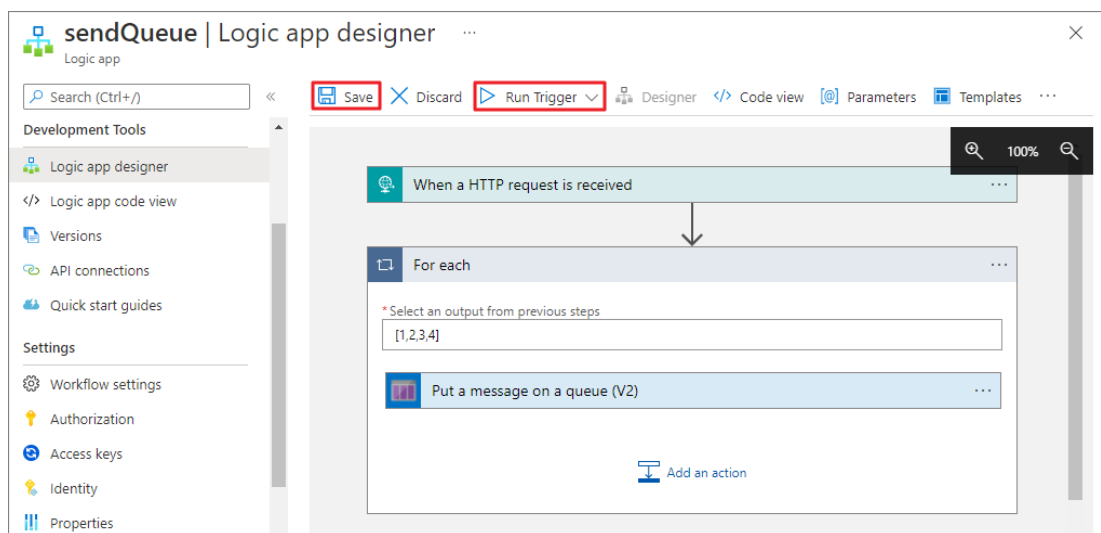
15. [For each] 타일에 간단한 배열(array) 값 `[1,2,3,4]`를 입력합니다.



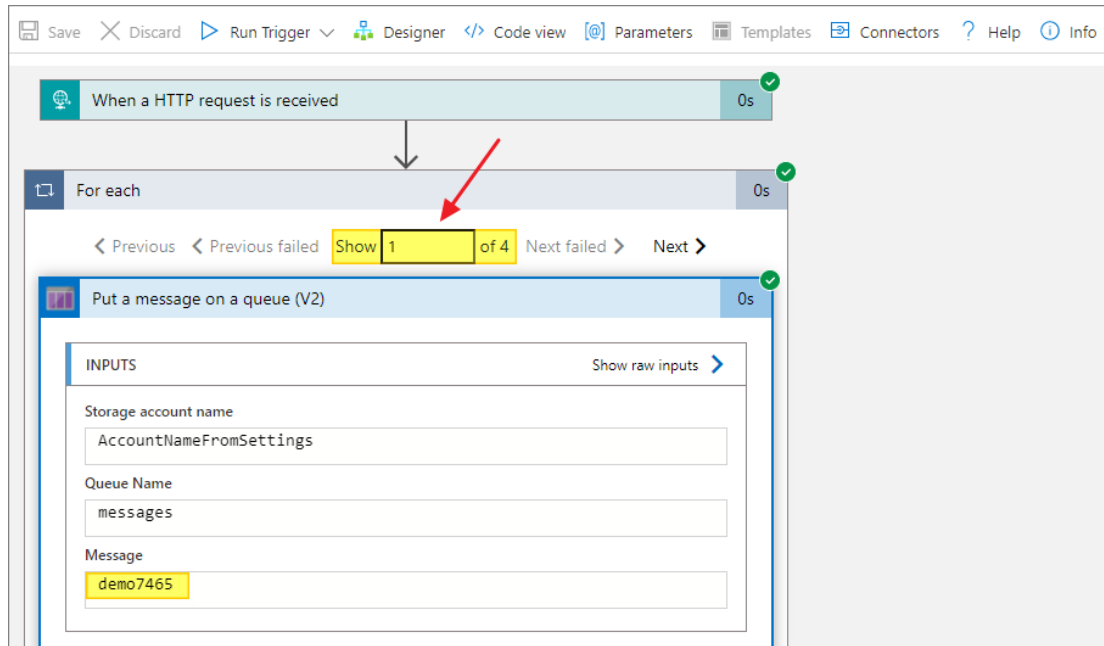
16. Logic App 디자이너에서 [Put a message on a queue (v2)] 타일을 드래그하여 [For each] 타일 안으로 드롭합니다.



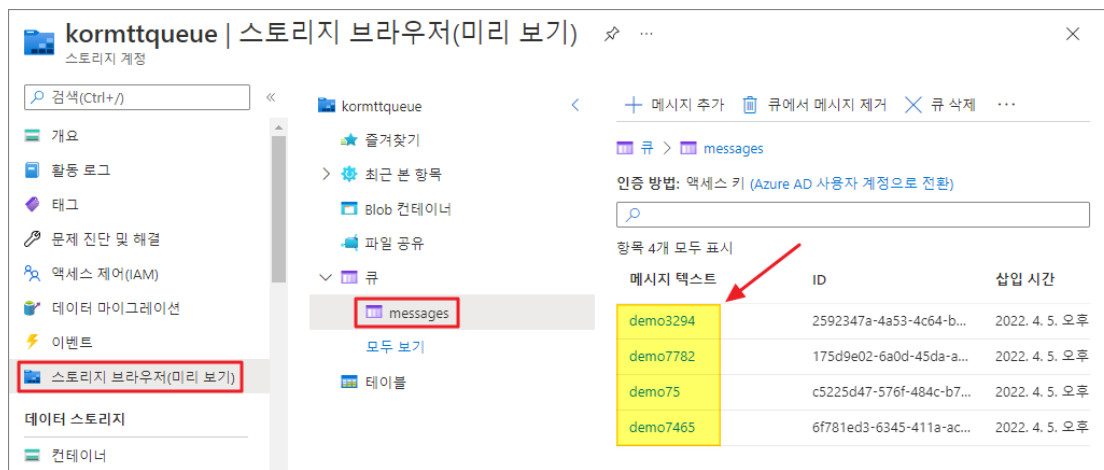
17. Logic App 디자이너의 메뉴에서 [Save]를 클릭한 후 [Run Trigger - Run]을 클릭합니다.



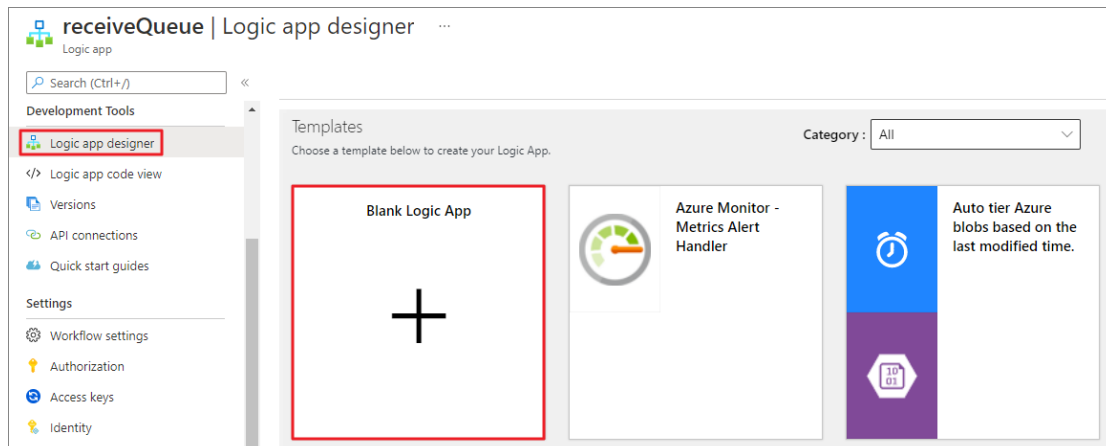
18. Logic App 실행 결과에서 4개의 배열 값이 실행된 것을 확인하고 임의로 생성된 Message 내용을 검토합니다.



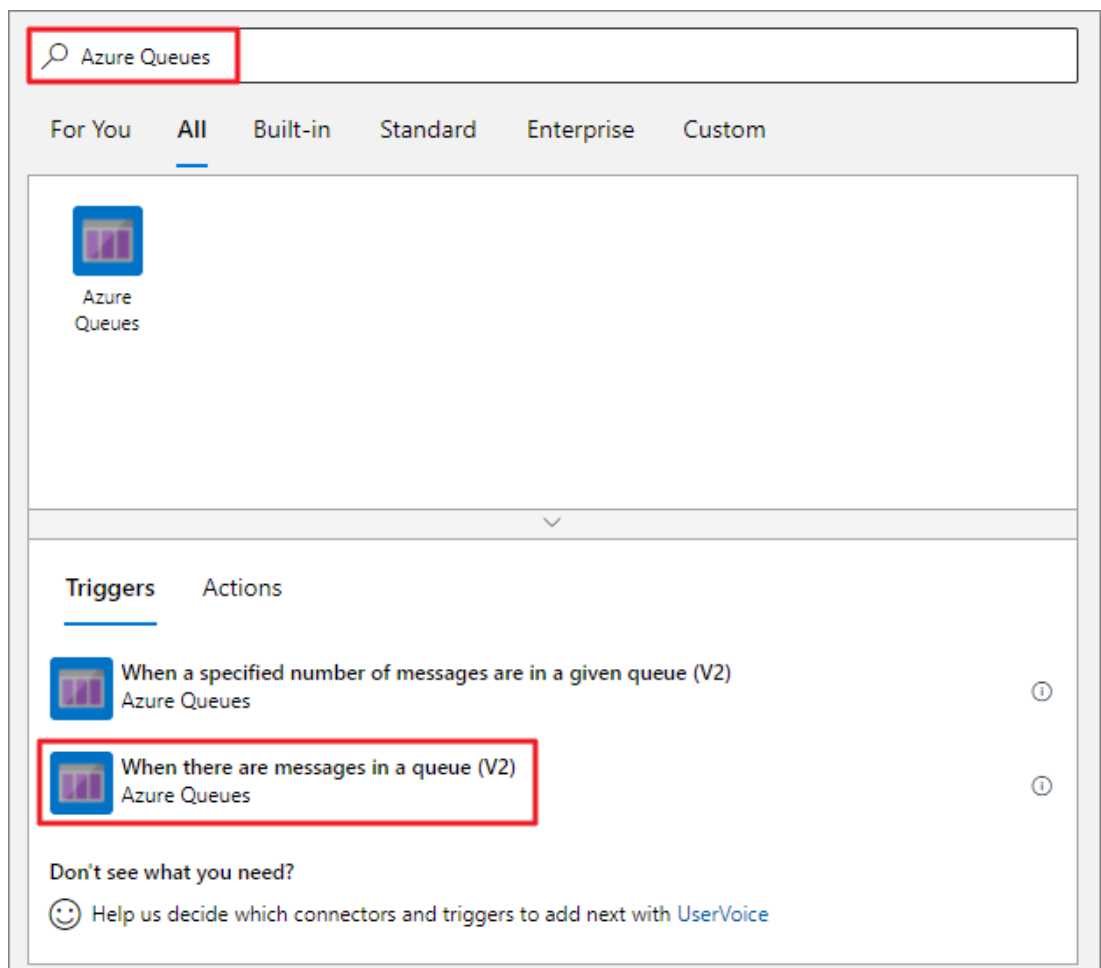
19. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동합니다. [큐 - messages]를 클릭한 후 Logic App을 통해 4개의 메시지가 수신된 것을 확인합니다.



20. 반복적으로 Storage Queue에 메시지를 출력하는 Logic App을 만들었습니다. 이제 두 번째 Logic App을 구성해야 합니다. 두 번째 Logic App은 첫 번째 Logic App의 출력을 기반으로 트리거합니다. [receiveQueue Logic app] 블레이드 [Development Tools - Logic app designer]로 이동한 후 [Blank Logic App] 타일을 클릭합니다.

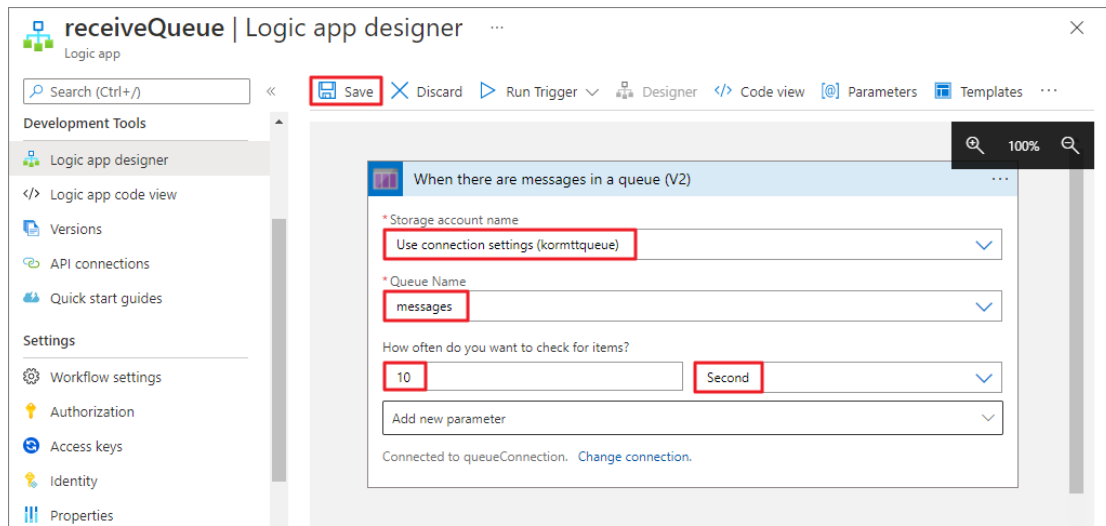


21. Logic App 디자이너의 검색창에 "Azure Queues"를 입력한 후 [Triggers] 탭에서 [When there are messages in a queue (V2)]를 클릭합니다.

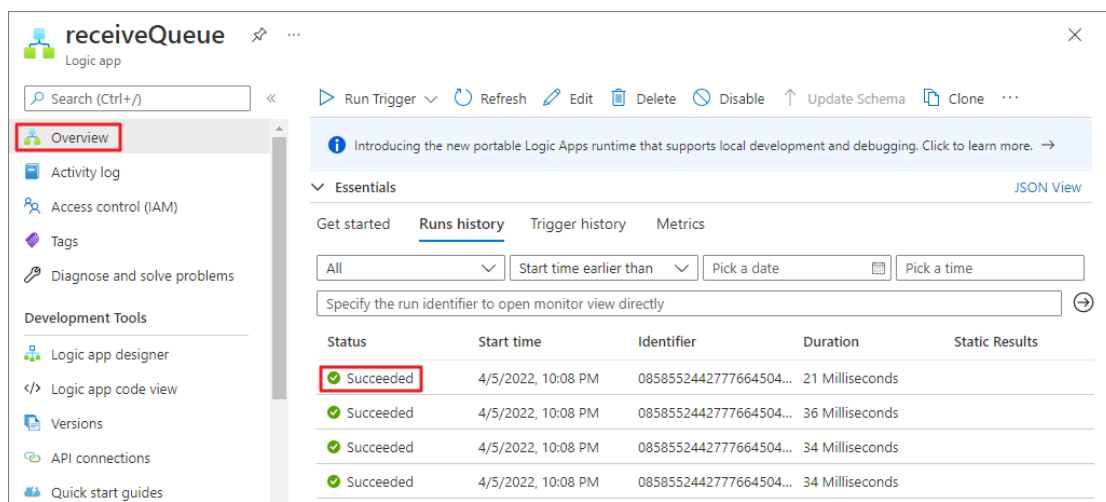


22. [When there are messages in a queue (V2)] 타일에서 아래와 같이 구성한 후 Logic App 디자이너의 메뉴에서 [Save]를 클릭합니다.

- Storage account name: 앞서 만들었던 스토리지 계정을 선택합니다.
- Queue Name: messages
- How often do you want to check for items?: 10초를 지정합니다.



23. [receiveQueue Logic app] 블레이드의 [Overview]로 이동한 후 메뉴에서 [Refresh]를 클릭합니다. [Runs history] 탭에 아래와 같이 sendQueue Logic App을 통해 Storage Queue로 보낸 4개의 메시지에 대한 작업이 표시되는 것을 확인합니다. 작업 중 하나를 클릭합니다.



24. [Logic app run]에서 아래와 같이 기존 Storage Queue에 있던 메시지를 가져온 것을 확인할 수 있습니다.

Logic app run ...
0858552442776645046486410267CU25

Run Details Resubmit Cancel Run Refresh Info

When there are messages in a queue (V2) 0s

INPUTS Show raw inputs >

Storage account name
Use connection settings (kormttqueue)

Queue Name
messages

OUTPUTS Show raw outputs >

Headers

Key	Value
x-ms-request-id	a3a0438f-3003-007b-3aee-48...
x-ms-client-request-id	7801cef2-069c-43be-8985-e0...
x-ms-version	2020-04-08

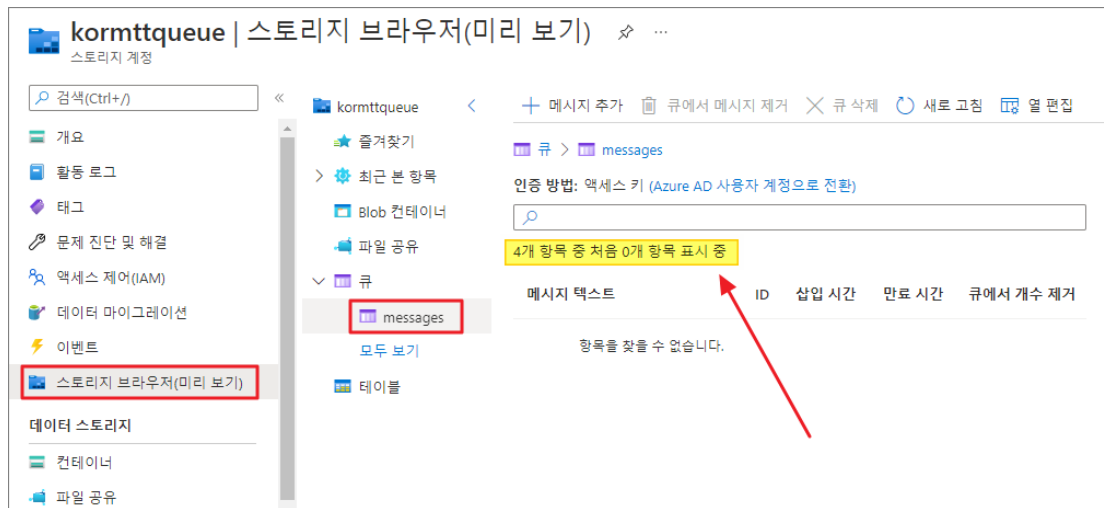
Body

```

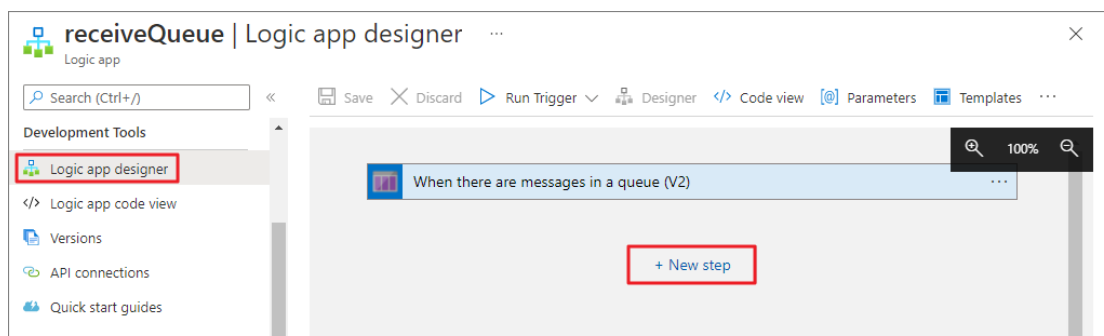
{
  "messageId": "C5Z25047-570T-484C-D7aZ-2T50950ZT283",
  "InsertionTime": "Tue, 05 Apr 2022 13:04:51 GMT",
  "ExpirationTime": "Tue, 12 Apr 2022 13:04:51 GMT",
  "PopReceipt": "AgAAAAMAAAAAAAvgBeT+5I2AE=",
  "TimeNextVisible": "Tue, 05 Apr 2022 13:08:57 GMT",
  "DequeueCount": "2",
  "MessageText": "demo75"
}
  
```

25. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동한 후 [큐 - messages]를 클릭합니다. 메뉴에서 [새로 고침]을 클릭하면 **receiveQueue** Logic App을 통해 처리된 4개의 메시지가 모두 사라진 것을 확인할 수 있습니다.

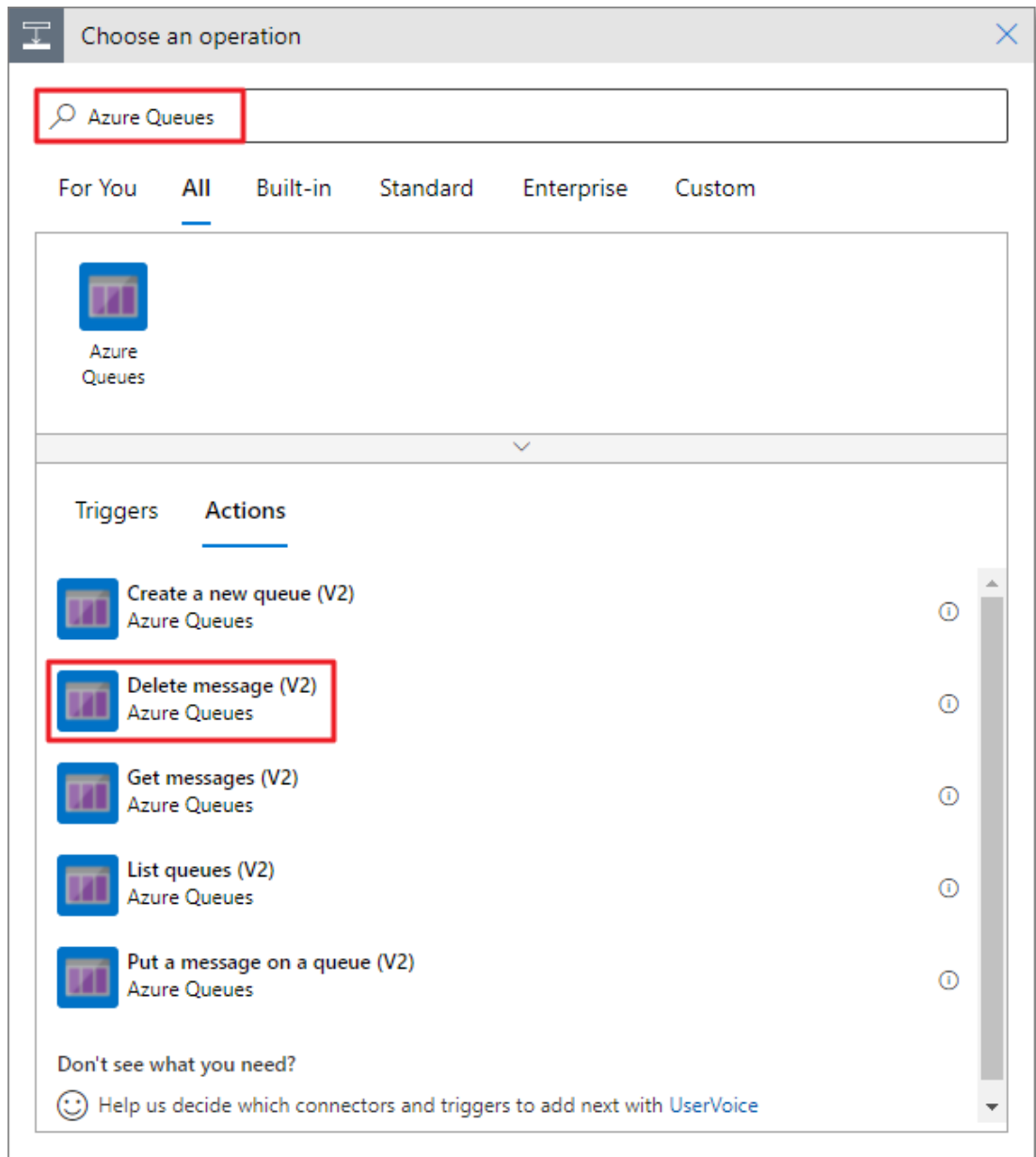
- "4개 항목 중 처음 0개 항목 표시 중"이라는 메시지가 표시됩니다.
- 따라서 메시지를 받은 후 실제로 메시지를 삭제하지 않았기 때문에 해당 메시지는 숨김 상태로 여전히 존재합니다.
- 실제 Storage Queue에서 메시지를 삭제하려면 **pop receipt**와 함께 삭제 요청을 전송해야 합니다.
- 그렇지 않은 경우 기본 가시성 타임아웃(default visibility timeout) 값인 30초 후에 Queue에 메시지가 다시 나타납니다.



26. 따라서 Logic App을 디자인할 때 수신한 메시지를 삭제하는 단계를 추가해야 합니다. [receiveQueue Logic app] 블레이드의 [Development Tools - Logic app designer]로 이동한 후 [New step]을 클릭합니다.

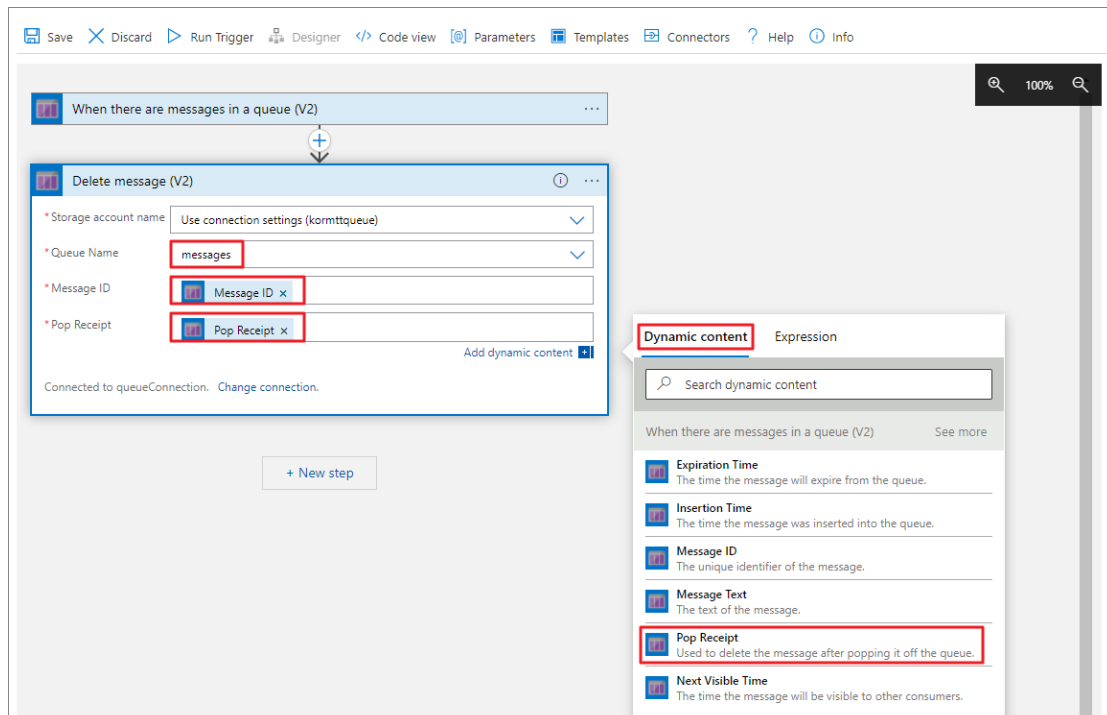


27. [Choose an operation] 타일에서 "Azure Queues"를 검색한 후 [Actions] 탭에서 [Delete message (V2)]를 클릭합니다.

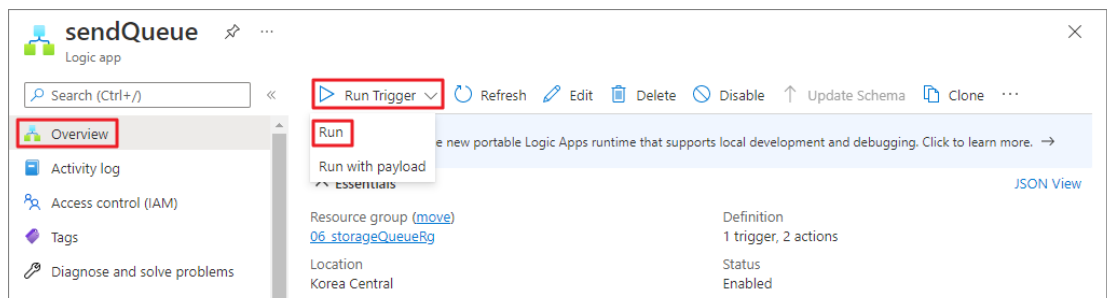


28. [Delete message (V2)] 타일에서 아래와 같이 구성한 후 Logic App 메뉴에서 [Save]를 클릭합니다.

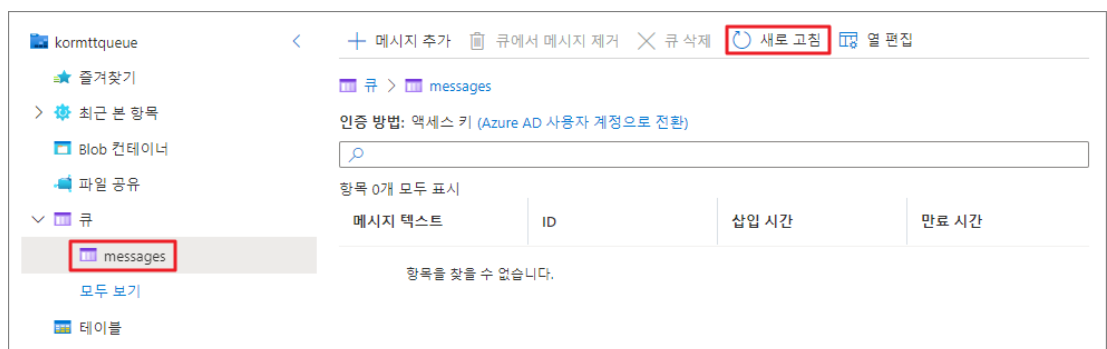
- Storage account name: 앞서 만들었던 스토리지 계정을 선택합니다.
- Queue Name: messages
- Message ID: "Add dynamic content"를 클릭한 후 [Dynamic content] 탭에서 "Message ID"를 클릭합니다.
- Pop Receipt: "Add dynamic content"를 클릭한 후 [Dynamic content] 탭에서 "Pop Receipt"를 클릭합니다.



29. 다시 Storage Queue에 메시지를 전송하기 위해 [sendQueue Logic app] 블레이드로 이동합니다. [Overview]의 메뉴에서 [Run Trigger - Run]을 클릭합니다.



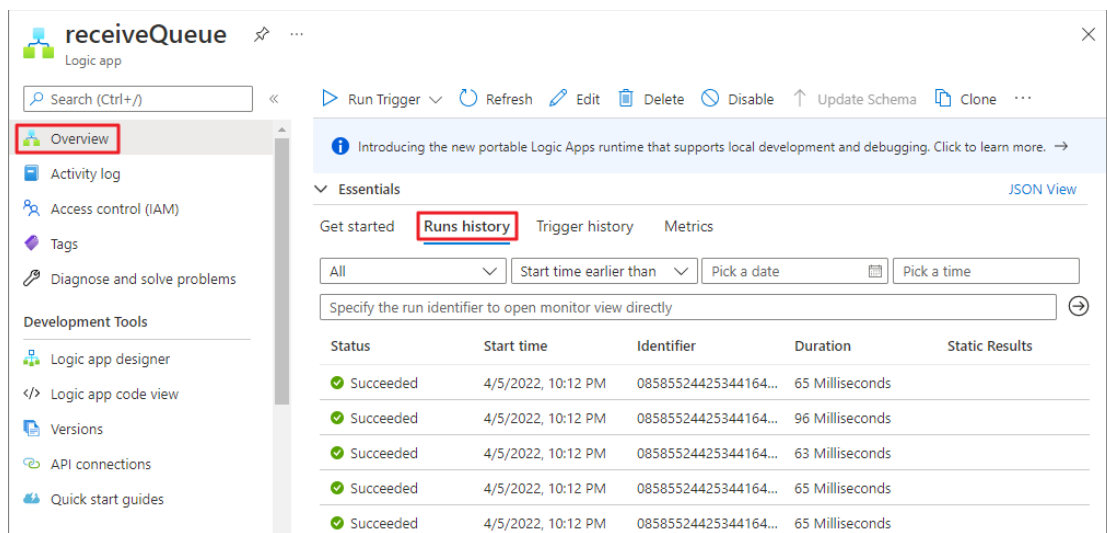
30. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동합니다. [큐 - messages]를 선택하고 메뉴에서 [새로 고침]을 클릭합니다. Storage Queue의 내용은 두 개의 Logic App을 통해 매우 빠르게 처리되기 때문에 실제 큐에서 메시지를 보기는 힘들지만 기존에 삭제되지 않았던 4개의 메시지와 새로 추가된 4개의 메시지가 Storage Queue에 들어오고 두 번째 Logic App을 통해 처리된 후 삭제된 것을 확인할 수 있습니다.



31. [receiveQueue Logic app] 블레이드의 [Overview]로 이동합니다. [Runs history] 탭에서 처리된 8개의

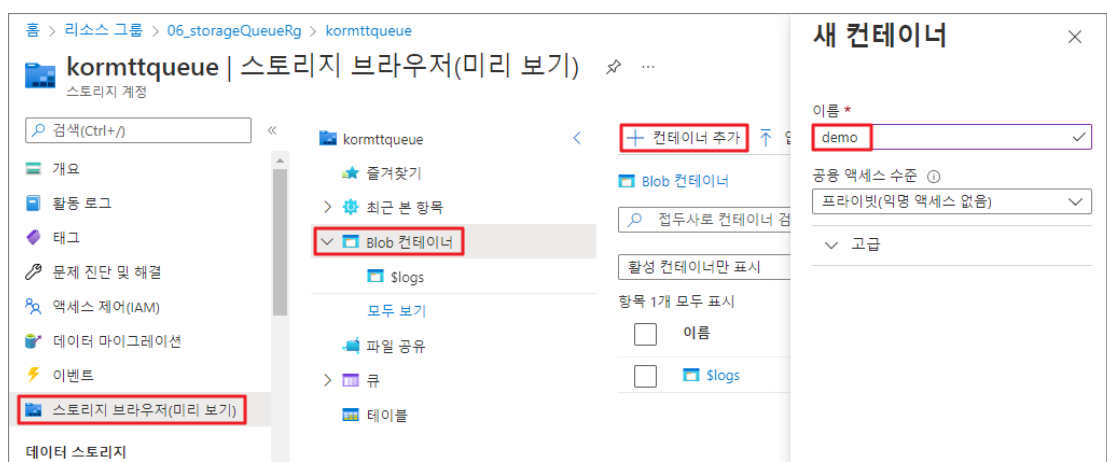
메시지가 표시되는 것을 확인할 수 있습니다.

- 이는 Storage Queue가 다른 서비스와 어떻게 작동하는지에 대해 매우 중요한 부분입니다.
- Storage Queue에서 처리된 메시지를 **Pop Receipt**와 함께 삭제하지 않으면 메시지는 숨겨진 상태에서 다시 표시되게 되어 여러 번 처리될 수 있습니다.
- 따라서 이러한 내용을 데이터베이스에 여러 번 중복적으로 추가하지 않도록 해야 하며 중복 메시지가 추가되는 경우 기존 엔터티와 병합할 수 있도록 해야 합니다.

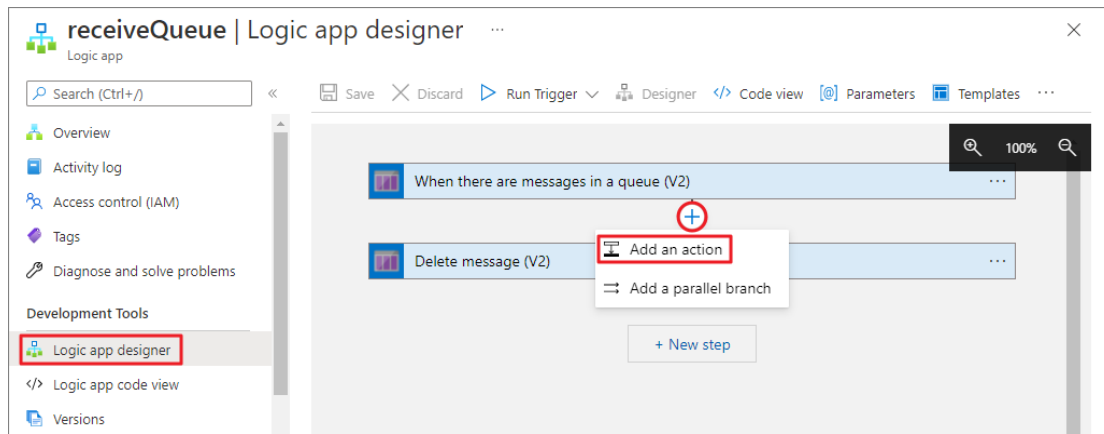


32. 마지막으로 기존 Logic App에 처리된 메시지를 Storage Blob에 저장하여 동시에 여러 메시지를 처리하고 파일로 출력하는 방법에 대해 실습합니다.

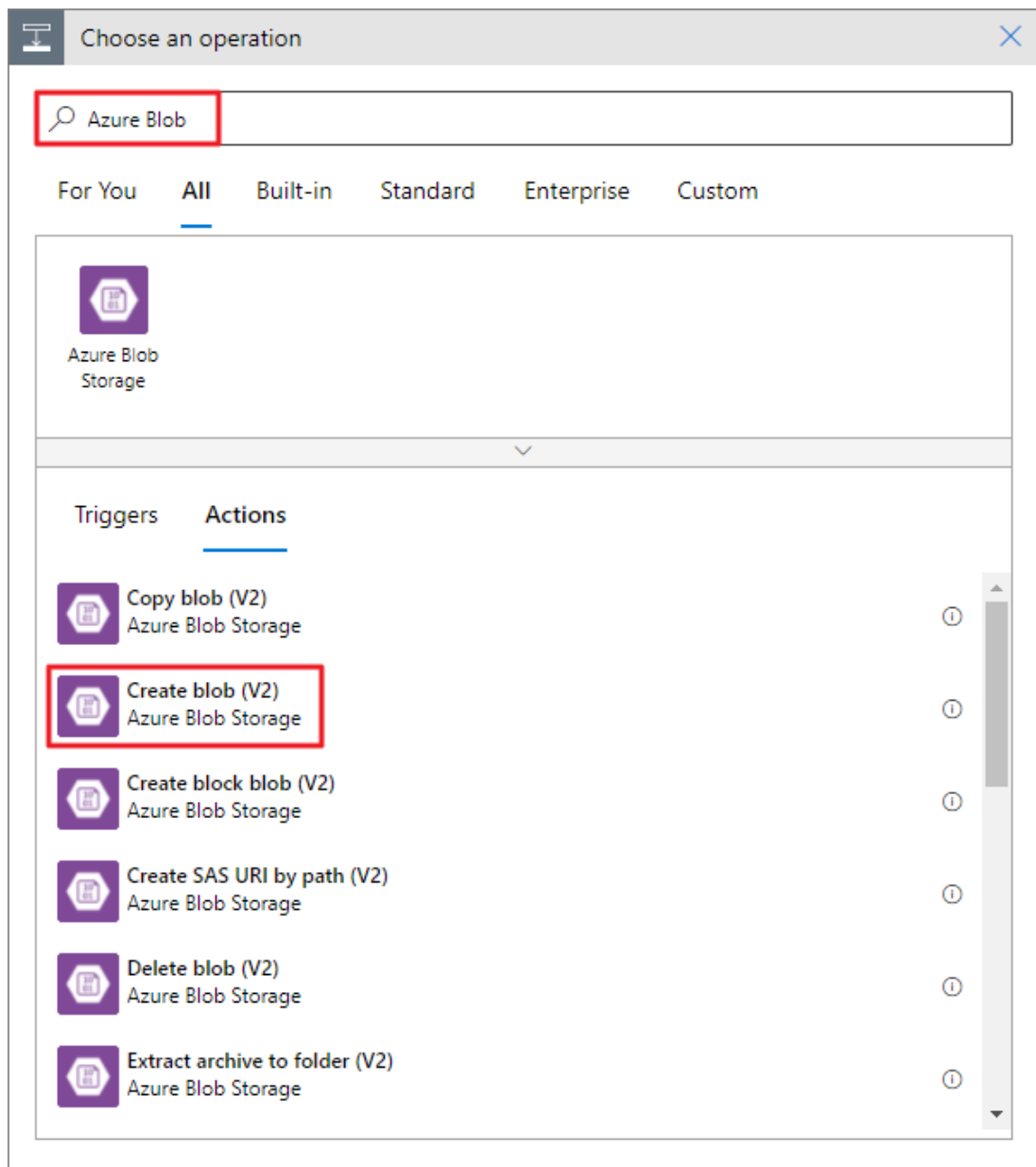
33. 브라우저에서 새 탭을 열고 Azure 포털로 이동합니다. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동합니다. [Blob 컨테이너]를 선택한 후 메뉴에서 [컨테이너 추가]를 클릭합니다. [새 컨테이너]에서 이름에 "demo"를 입력한 후 [만들기]를 클릭합니다.



34. [receiveQueue Logic app] 블레이드의 [Development Tools - Logic app designer]로 이동합니다. Logic App 디자이너에서 [When there are messages in a queue (V2)] 타일 아래의 [+ - Add an action]을 클릭합니다.



35. [Choose an operation] 타일의 검색창에 "Azure Blob"을 검색한 후 [Actions] 탭에서 [Create blob (V2)]를 클릭합니다.



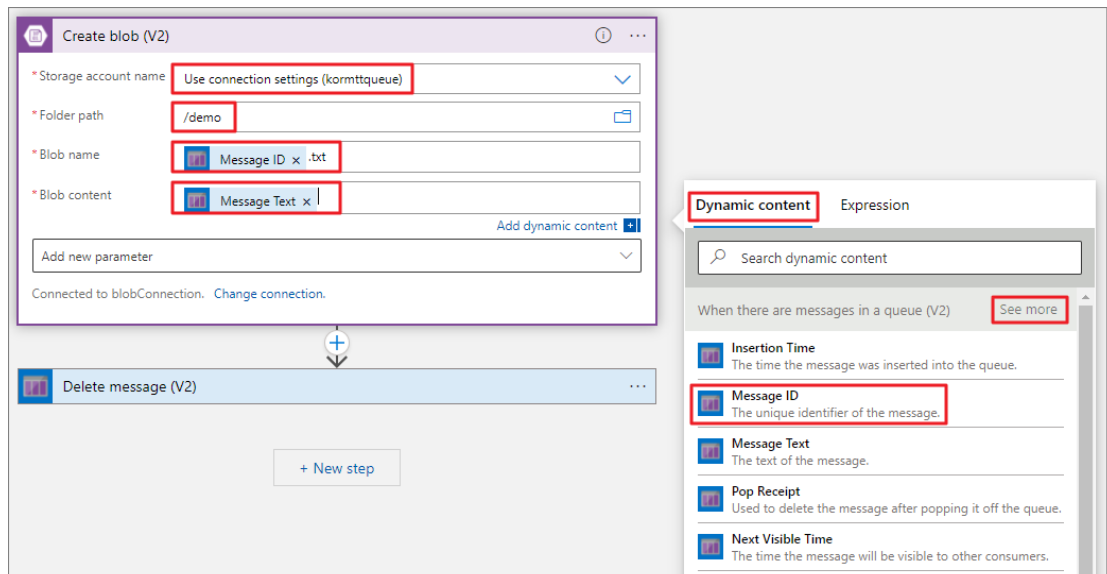
36. [Azure Blob Storage] 타일에서 아래와 같이 구성한 후 [Create]를 클릭합니다.

- Connection name: blobConnection
- Authentication type: Access Key
- Azure Storage Account name: 앞서 만들었던 스토리지 계정 이름을 붙여 넣습니다.
- Azure Storage Account Access Key: 앞서 만들었던 스토리지 계정의 **key1** 값을 붙여 넣습니다.

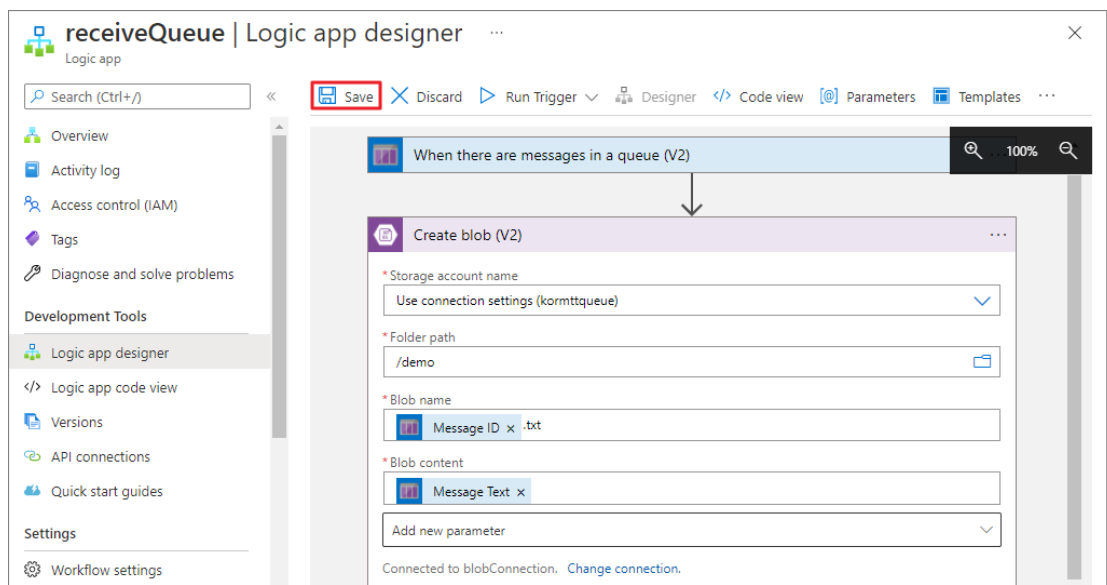
The screenshot shows the 'Azure Blob Storage' configuration window. It has four input fields, each with an information icon (i) to its left. The first field is 'Connection name' with the value 'blobConnection'. The second is 'Authentication type' with a dropdown menu showing 'Access Key'. The third is 'Azure Storage Account name' with the value 'kormttqueue'. The fourth is 'Azure Storage Account Access Key' with a masked value represented by dots. At the bottom center is a blue 'Create' button.

37. [receiveQueue Logic app] 블레이드의 Logic App 디자이너 탭으로 전환합니다. [Create blob (V2)] 타일에서 아래와 같이 구성합니다.

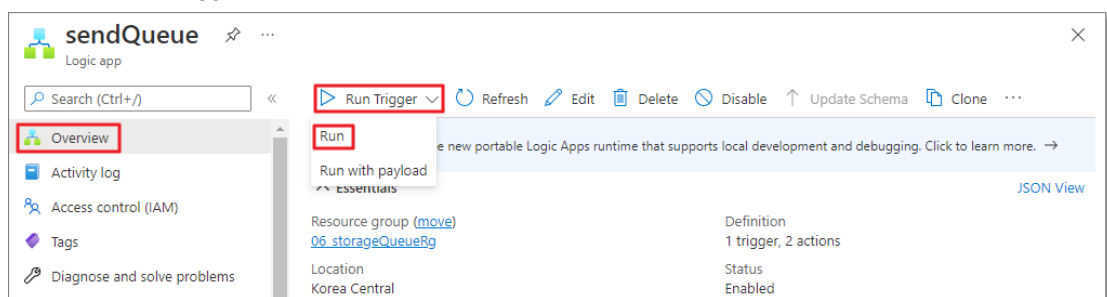
- Storage account name: 앞서 만들었던 스토리지 계정을 선택합니다.
- Folder path: [Show picker] 아이콘을 클릭한 후 위에서 만들었던 **demo** 컨테이너를 선택합니다.
- Blob name: "Add dynamic content"를 클릭한 후 [Dynamic content] 탭에서 "Message ID"를 선택합니다. 추가된 동적 콘텐츠 뒤에 **.txt** 값을 입력합니다.
- Blob content: "Add dynamic content"를 클릭한 후 [Dynamic content] 탭에서 "See more"를 클릭합니다. [Dynamic content] 탭에서 "Message Text"를 선택합니다.



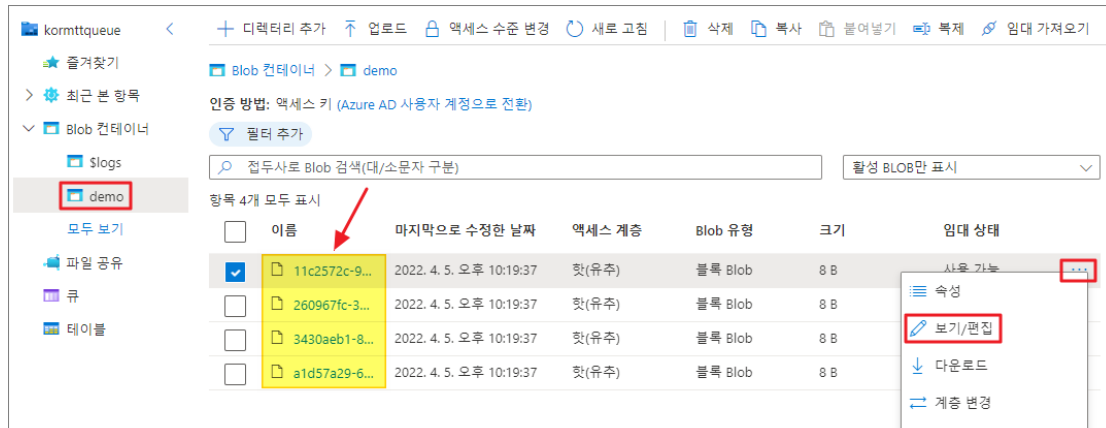
38. Logic App 디자이너의 메뉴에서 [Save]를 클릭합니다.



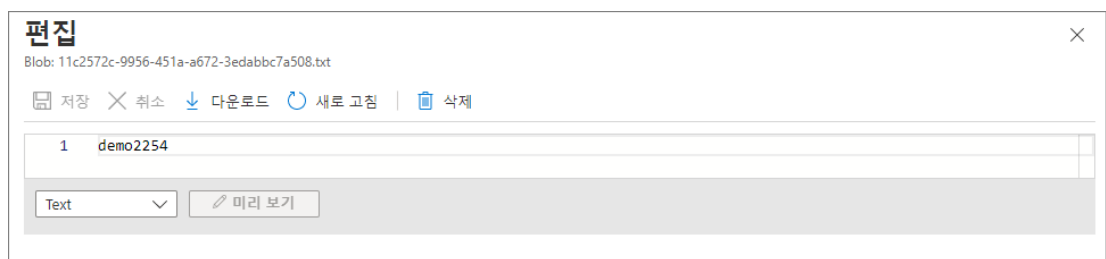
39. 이제 마지막 테스트를 진행할 순서입니다. [sendQueue Logic app] 블레이드의 [Overview]로 이동합니다. 메뉴에서 [Run Trigger - Run]을 클릭합니다.



40. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동합니다. [Blob 컨테이너 - demo]를 선택한 후 메뉴에서 [새로 고침]을 클릭합니다. Logic App 실행 간격으로 지정한 10초 후에 4개의 파일이 생성된 것을 확인할 수 있습니다. 파일 중 하나를 선택한 후 [... - 보기/편집]을 클릭합니다.

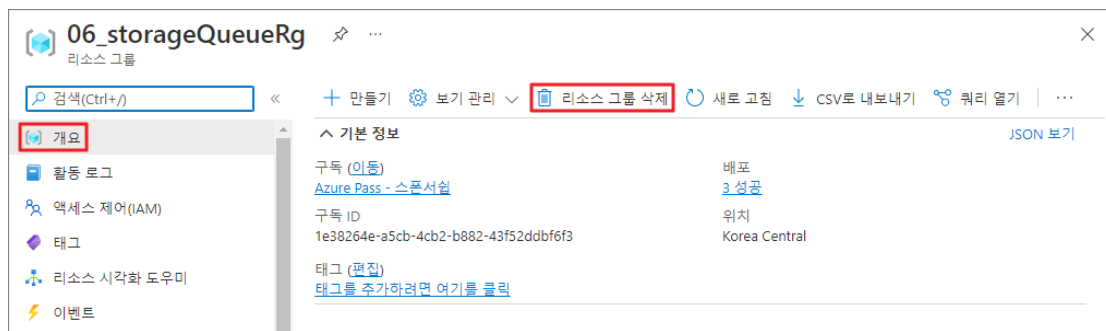


41. 아래와 같이 파일 내용에 메시지의 콘텐츠가 표시되는 것을 확인할 수 있습니다.





TASK 04. 리소스 정리

1. Azure 포털에서 [06_storageQueueRg 리소스 그룹] 블레이드로 이동한 후 메뉴에서 [리소스 그룹 삭제]를 클릭합니다.



2. 리소스 그룹 삭제 확인 창에서 리소스 그룹 이름을 입력한 후 [삭제]를 클릭합니다.



 "06_storageQueueRg"을(를) 삭제하시겠... ×



경고! "06_storageQueueRg" 리소스 그룹을 삭제하면 되돌릴 수 없습니다. 지금 수행하려는 작업은 취소할 수 없습니다. 계속 진행하면 이 리소스 그룹과 그 안의 모든 리소스가 영구적으로 삭제됩니다.

리소스 그룹 이름 입력:
 ✓

관련 리소스
이 리소스 그룹에 있는 5개의 리소스가 삭제됩니다.

이름	형식	위치
 azureblob	API 연결	Korea Central
 azurequeues	API 연결	Korea Central
 kormttqueue	스토리지 계정	Korea Central
 receiveQueue	논리 앱	Korea Central
 sendQueue	논리 앱	Korea Central