

한국 마이크로소프트

Microsoft Technical Trainer

Enterprise Skills Initiative

Microsoft Azure

Azure Functions

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
2022.02.07	0.6.0	우진환	TASK 01 ~ TASK 02 작성
2022.02.08	1.0.0	우진환	TASK 03 작성
2022.04.01	1.1.0	우진환	TASK 04 작성, Azure 포털 UI 변경 적용

목차

주요 기능.....	5
FUNCTION APPS 작동 방법	5
트리거(TRIGGER)와 바인딩(BINDING).....	7
TASK 01. FUNCTION APPS 만들기	8
TASK 02. AZURE 포털에서 코딩.....	15
TASK 03. VISUAL STUDIO CODE에서 개발.....	17
TASK 04. 리소스 정리.....	27

이 실습에서는 서버나 인프라를 규모에 맞게 관리하지 않고 코드로만 관리하는 serverless 서비스인 Azure Functions에 대해 알아봅니다.

Azure Functions는 작은 코드 조각(혹은 functions)을 실행하기 위한 서비스이며, .NET이나 Java 혹은 지원되는 다른 언어에서 작성한 코드의 기능을 호출할 수 있습니다. Azure Functions는 serverless를 실행하기 위한 매우 저렴하며 확장 가능한 서비스입니다. 즉 Azure Functions를 사용하기 위해 별도로 관리할 인프라는 없습니다.

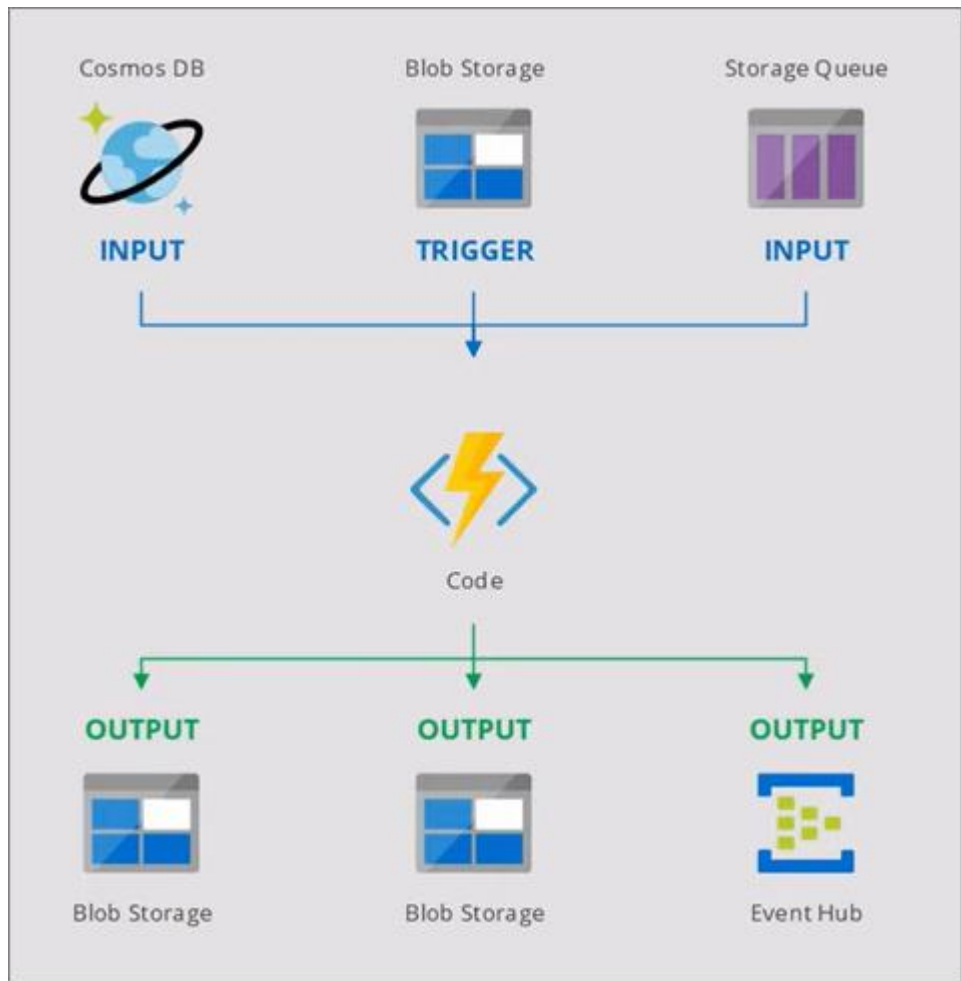
주요 기능

Azure Functions는 다음과 같은 기능을 제공합니다.

- C# (.NET 혹은 .NET Core), Java, JavaScript (NodeJS), Python, F#, PowerShell, TypeScript 언어를 모두 지원합니다.
- 매월 400,000 실행 시간, 1,000,000 번 실행이 무료이므로 대부분의 경우 무료로 Functions를 사용할 수 있습니다. 이 이상을 사용하는 경우라도 비용은 매우 저렴합니다.
- NPM 혹은 NuGet을 사용하여 고유한 종속성을 가져올 수 있습니다.
- App Service에서 제공하는 통합 보안을 사용할 수 있으며 Azure AD 혹은 기타 외부 공급자와 통합할 수 있습니다.
- 외부 도구와 결합할 수 있으므로 간단히 통합 작업을 진행할 수 있습니다.
- Azure 포털, Visual Studio Code, 다른 개발 도구를 통해 유연한 개발이 가능합니다. 또한 GitHub, Azure DevOps 및 기타 외부 리포지토리와 통합되어 있습니다.
- Azure Functions 용 SDK 전체가 오픈 소스로 제공됩니다.

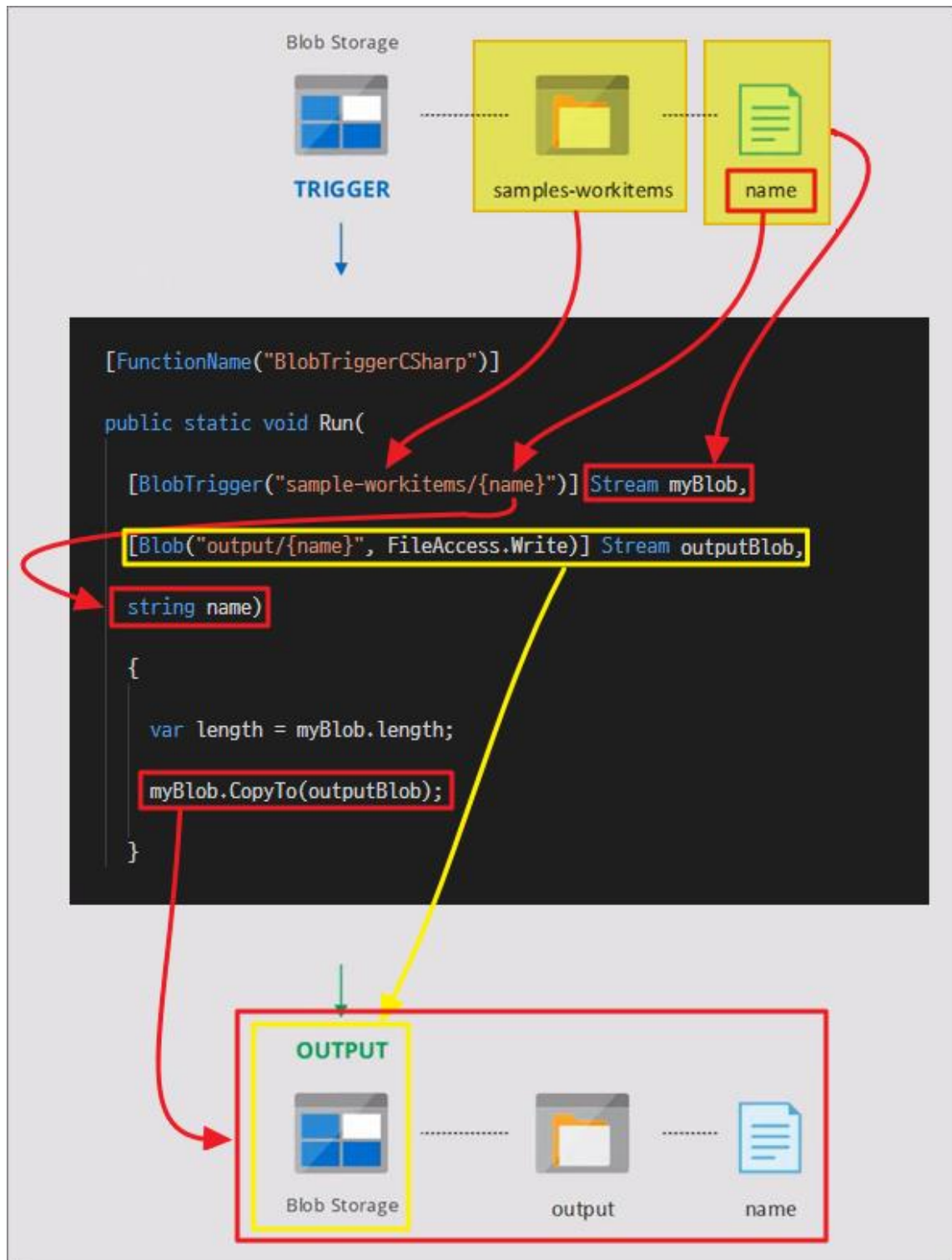
Function Apps 작동 방법

Azure Functions는 코드의 조각이며 실행하기 위해서는 코드 조각을 실행하려는 트리거가 필요합니다. 트리거는 코드와 관련된 작업이며 실행을 위해 코드에 몇 가지 기본 정보를 전달합니다. 물론 코드가 완료되면 일종의 출력의 있습니다. 따라서 트리거 입력(input) 및 출력(output)에 대한 바인딩은 매우 간단합니다. 예를 들어 Storage Blob에 바인딩하여 blob에 새 파일이 추가되면 코드를 트리거하고 해당 파일을 처리한 다음 blob에 다시 넣을 수 있습니다. 트리거는 하나만 사용할 수 있지만 입력과 출력은 하나로 제한되지 않습니다. 예를 들어 Storage Blob을 처리하는 동안 Cosmos DB에서 추가 데이터를 가져오거나 Storage Queue에서 추가 정보를 가져올 수 있습니다. 또한 여러 출력을 가질 수 있습니다. 따라서 파일을 처리한 후 Queue에서 일부 정보를 가져와 Storage Blob으로 출력할 수 있습니다. 혹은 다른 Storage Blob에 추가 로깅을 저장하고 Event Hub에서 완료된 처리를 다른 애플리케이션이 알 수 있도록 일부 이벤트를 넣을 수도 있습니다. 이러한 모든 통합은 매우 쉽습니다.



아래와 같이 간단한 예제를 확인해 보도록 하겠습니다. 이 예제에서는 Storage Blob에서 데이터를 처리하여 Storage Blob으로 출력합니다. 따라서 이는 blob에서 파일을 트리거로 가져오는 코드입니다.

- **BlobTrigger**: blob 트리거가 있으며 매개 변수를 전달하고 **BlobTrigger**를 지정하는 방식으로 수행합니다. 이는 Storage Blob에 새 파일이 생길 때마다 트리거한다는 것을 Function에 알려 줍니다.
- **sample-workitems**: 이 이름의 컨테이너를 지정합니다. 따라서 Function은 이 컨테이너에 새 파일이 생기거나 수정될 때마다 트리거됩니다.
- **{name}**: 표현식이며 이를 사용하여 처리 중인 파일의 이름을 가져옵니다. 이 표현식을 동일하게 사용하여 동일한 **string name** 변수에 전달할 수 있습니다.
- **Stream myBlob**: 이 트리거에 대해 파일 내용을 넣을 변수를 정의합니다. 이 경우 스트림이므로 파일과 모든 내용을 **myBlob** 변수에 넣습니다.
- **Blob("output/{name}", FileAccess.Write)] Stream outputBlob**: 출력을 위해 Blob을 추가합니다. 그 뒤에 "컨테이너/이름"을 지정하고 파일을 처리하고 있기 때문에 **FileAccess**를 지정합니다.
- **myBlob.CopyTo(outputBlob)**: blob의 복사본을 추가 스토리지의 추가 컨테이너에 복사합니다.



트리거(Trigger)와 바인딩(Binding)

트리거는 Function을 실행하는 원인입니다. 트리거는 Function이 호출되는 방식을 정의하며 Function에는 정확히 하나의 트리거만 있어야 합니다. 트리거는 종종 Function의 페이로드로 제공되는 관련 데이터가

있습니다.

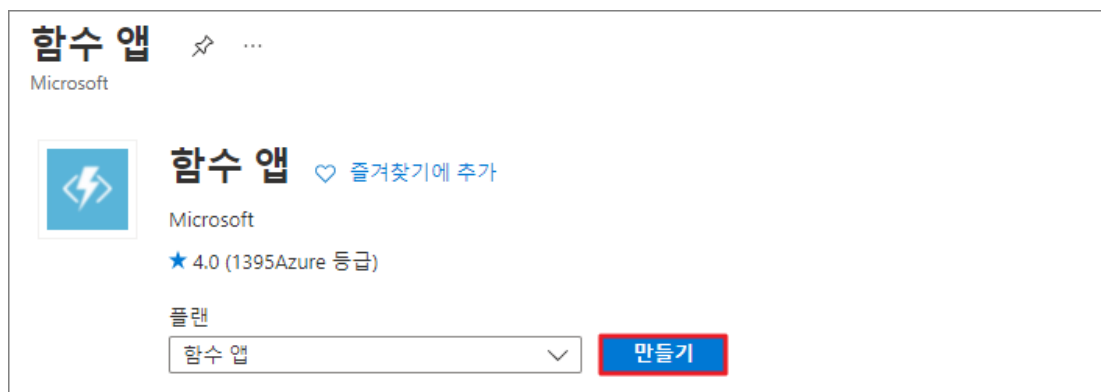
Function에 바인딩하는 것은 다른 리소스를 Function에 선언적으로 연결하는 방법입니다. 바인딩은 입력 바인딩(input binding), 출력 바인딩(output binding) 혹은 둘 다 연결될 수 있습니다. 바인딩의 데이터는 매개 변수로 Function에 전달됩니다. 필요에 따라 다른 바인딩을 mix & match 할 수 있습니다. 바인딩은 선택 사항이며 Function에는 하나 이상의 입력 및/혹은 출력 바인딩이 있을 수 있습니다.

지원되는 바인딩과 트리거는 아래 표를 참고할 수 있습니다.

유형	1.x	2.x 이상	Trigger	Input	Output
Blob storage	✓	✓	✓	✓	✓
Azure Cosmos DB	✓	✓	✓	✓	✓
Azure SQL (preview)		✓		✓	✓
Dapr		✓	✓	✓	✓
Event Grid	✓	✓	✓		✓
Event Hubs	✓	✓	✓		✓
HTTP & webhooks	✓	✓	✓		✓
IoT Hub	✓	✓	✓		✓
Kafka		✓	✓		✓
Mobile Apps	✓			✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓	✓		✓
RabbitMQ		✓	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
SignalR		✓		✓	✓
Table storage	✓	✓		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓

TASK 01. Function Apps 만들기

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



2. [함수 앱 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [다음]을 클릭합니다.
 - [프로젝트 세부 정보 - 리소스 그룹]: "새로 만들기"를 클릭한 후 "02_functionAppsRg"를 입력합니다.

- [인스턴스 정보 - 함수 앱 이름]: 중복되지 않는 고유한 이름을 입력합니다.
- [인스턴스 정보 - 게시]: 코드
- [인스턴스 정보 - 런타임 스택]: .NET
- [인스턴스 정보 - 버전]: 3.1
- [인스턴스 정보 - 지역]: Korea Central

함수 앱 만들기 ...

기본 호스팅 네트워킹(미리 보기) 모니터링 태그 검토 + 만들기

함수 앱을 만들면 함수를 논리 단위로 그룹화하여 리소스를 더 쉽게 관리, 배포 및 공유할 수 있습니다. 함수를 사용하면 먼저 VM을 만들거나 웹 애플리케이션을 게시하지 않고도 서비스 환경에서 코드를 실행할 수 있습니다.

프로젝트 세부 정보

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

구독 * ① Azure Pass - 스폰서쉽

리소스 그룹 * ① (신규) 02_functionAppRg
새로 만들기

인스턴스 정보

함수 앱 이름 * kormttfuncapp .azurewebsites.net

게시 * ☒ 코드 ☐ Docker 컨테이너

런타임 스택 * .NET

버전 * 3.1

지역 * Korea Central

3. [호스팅] 탭에서 아래와 같이 구성한 후 [다음]을 클릭합니다.

- 스토리지 계정: "새로 만들기"를 클릭한 후 중복되지 않는 고유한 스토리지 계정 이름을 입력합니다. Function은 별도의 서버가 없기 때문에 코드를 넣고 저장할 스토리지가 필요합니다. Function 코드를 포함하여 Function을 수행하는 모든 것이 스토리지에 저장됩니다. Function의 전체 코드 기반과 Function의 상태가 스토리지에 저장되며 새로운 서버가 초기화될 때마다 이 스토리지에서 코드를 가져옵니다.
- 운영 체제: Windows
- 플랜 유형: "사용량(서버리스)"를 선택합니다. 사용량 플랜은 serverless이므로 별도의 서버를 관리하지 않습니다. Azure는 Function이 실행될 때마다 서버를 제공하고 실행되지 않을 때 해당 서버를 제거합니다. Azure는 0~200대의 서버를 필요에 따라 결정합니다. 즉 많은 데이터를 처리하려는 경우 Azure는 계속해서 추가 서버를 제공하고 데이터는 더 빠르게 처리됩니다.

함수 앱 만들기 ...

기본 호스팅 네트워킹(미리 보기) 모니터링 태그 검토 + 만들기

스토리지

함수 앱을 만들 때 Blob, 큐 및 테이블 스토리지를 지원하는 범용 Azure Storage 계정을 만들거나 연결해야 합니다.

스토리지 계정 * (신규) kormttfuncstorage ▼
[새로 만들기](#)

운영 체제

런타임 스택 선택에 따라 운영 체제가 권장되었습니다.

운영 체제 * ☐ Linux ☒ Windows

계획

선택하는 플랜에 따라 앱 스케일링 방법, 사용할 수 있는 기능 및 가격 책정 방법이 결정됩니다. [자세히](#)

플랜 유형 * ① 사용량(서버리스) ▼

4. [Networking] 탭에서 [다음]을 클릭합니다. 이전 페이지에서 플랜 유형을 "Functions 프리미엄" 혹은 "App Service 요금제"의 Standard 이상을 선택한 경우 사용할 수 있으며 App Service와 마찬가지로 가상 네트워크 통합, 가상 네트워크 트리거, 하이브리드 연결과 같은 기능을 사용할 수 있습니다.

함수 앱 만들기 ...

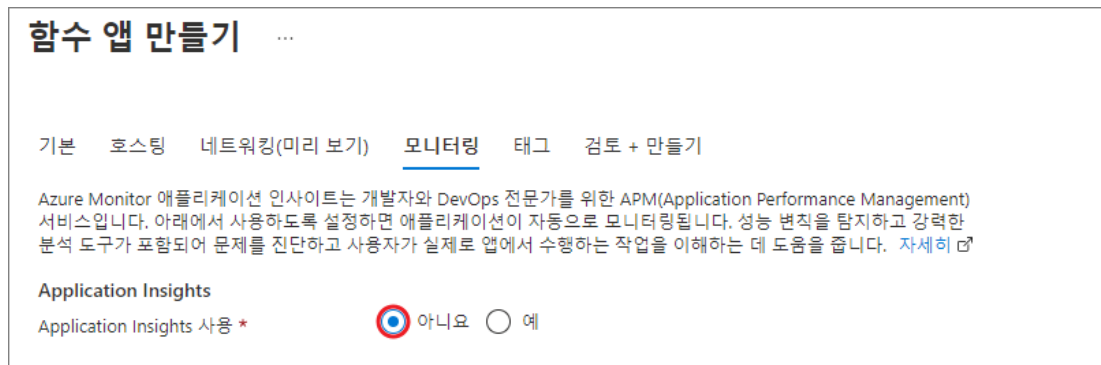
기본 호스팅 네트워킹(미리 보기) 모니터링 태그 검토 + 만들기

인터넷에 공개되거나 Azure 가상 네트워크로 격리되는 인바운드 주소를 사용하여 기능 애플(들) 프로비전할 수 있습니다. 아웃바운드 트래픽으로 기능 애플(들) 프로비전할 수도 있습니다. 그러면 가상 네트워크의 엔드포인트에 연결하고, 네트워크 보안 그룹으로 관리하거나, 가상 네트워크 경로의 영향을 받을 수 있습니다. 기본적으로 앱은 인터넷에 공개되며 가상 네트워크에 연결할 수 없습니다. 이러한 측면은 앱을 프로비전한 후에 변경할 수도 있습니다. [자세히](#)

⚠ 네트워크 삽입은 Functions Premium과 Standard, Premium, Premium V2, Premium V3 Dedicated App Service 요금제에 서만 사용할 수 있습니다.

네트워크 삽입 사용 ☐ 켜기 ☒ 끄기

5. [모니터링] 탭에서 Application Insights 사용을 "아니요"로 선택한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를 클릭합니다.



6. 새로 만든 [함수 앱] 블레이드의 [개요]로 이동합니다. Function에 대한 URL, 요금제, 런타임 버전 등의 정보를 확인할 수 있습니다.



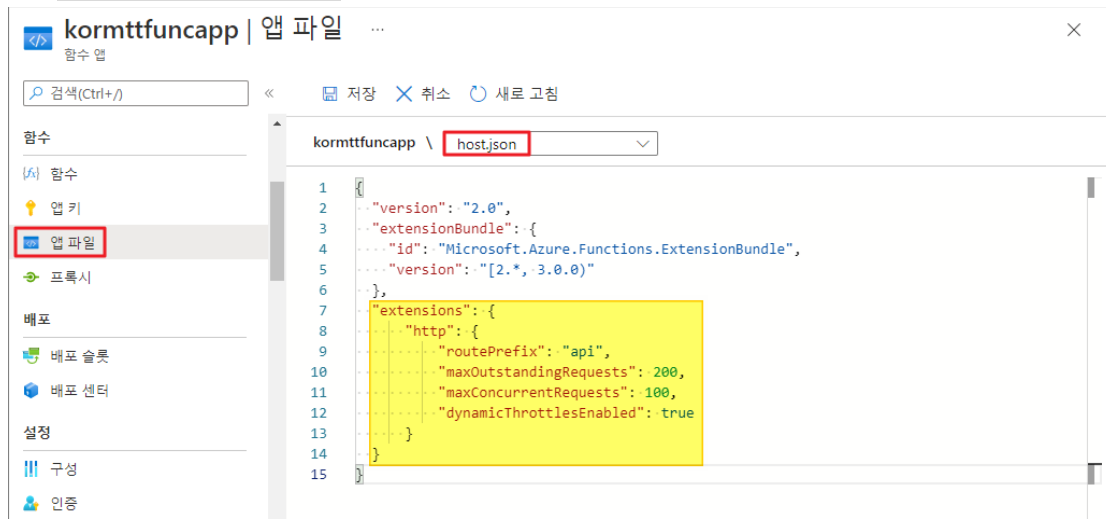
7. [함수 앱] 블레이드의 [함수 - 앱 키]로 이동합니다. 모든 HTTP Function에 액세스하기 위한 호스트 키 설정과 시스템 키 설정을 확인할 수 있습니다.



8. [함수 앱] 블레이드의 [함수 - 앱 파일]로 이동합니다. `host.json` 파일의 내용이 표시되며 Function Apps이 어떤 환경으로 실행되는지 설정할 수 있습니다. `host.json` 메타데이터 파일은 Function Apps의 모든 Function에 적용되는 전역 구성 옵션을 포함하고 있는 파일입니다. 예를 들어 `host.json` 파일에

HTTP 섹션을 아래와 같이 적용할 수 있습니다.

- HTTP 섹션은 HTTP 트리거가 작동하는 방식을 제어하는 섹션입니다.
- `routePrefix`: 라우팅에 대한 접두사기 `/api`라는 경로를 정의합니다.
- `maxOutstandingRequests`: 요청이 서비스에서 `TooBusy` 응답을 받기 전에 대기할 수 있는 요청 수를 지정합니다.
- `maxConcurrentRequests`: 해당 시간에 발생할 수 있는 Function에 대한 병렬 실행 수입니다.



9. [함수 앱] 블레이드의 [설정 - 구성]으로 이동한 후 [함수 런타임 설정] 탭을 클릭합니다. Function의 런타임 버전과 일일 사용 할당량을 이 페이지에서 확인할 수 있습니다. [일반 설정] 탭에서는 플랫폼 설정과 디버깅, 클라이언트 인증서 설정을 확인할 수 있습니다.



10. Azure 포털의 [함수 앱] 블레이드에서 [함수 - 함수]로 이동한 후 [만들기]를 클릭합니다.



11. [함수 만들기]에서 아래와 같이 구성한 후 [만들기]를 클릭합니다.
 - 개발 환경: "포털에서 개발". 선택한 개발 환경에 따라 필요한 지침을 안내합니다.
 - 템플릿 선택: "HTTP trigger". 기본 템플릿으로 사용할 수 있는 여러 템플릿이 표시됩니다.
 - [템플릿 세부 정보 - 새 함수]: HttpTriggerFunctionDemo
 - [템플릿 세부 정보 - Authorization level]: Function

함수 만들기

개발 환경 선택

지침은 개발 환경에 따라 달라집니다. [자세한 정보](#)

개발 환경

포털에서 개발

템플릿 선택

템플릿을 사용하여 함수를 만듭니다. 트리거는 함수를 호출하는 이벤트 유형을 설명합니다. [자세한 정보](#)

필터

템플릿	설명
HTTP trigger	A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string
Timer trigger	A function that will be run on a specified schedule
Azure Queue Storage trigger	A function that will be run whenever a message is added to a specified Azure Storage queue
Azure Service Bus Queue trigger	A function that will be run whenever a message is added to a specified Service Bus queue
Azure Service Bus Topic trigger	A function that will be run whenever a message is added to the specified Service Bus topic
Azure Blob Storage trigger	A function that will be run whenever a blob is added to a specified container
Azure Event Hub trigger	A function that will be run whenever an event hub receives a new event

템플릿 세부 정보

HTTP trigger 함수를 만들려면 추가 정보가 필요합니다. [자세한 정보](#)

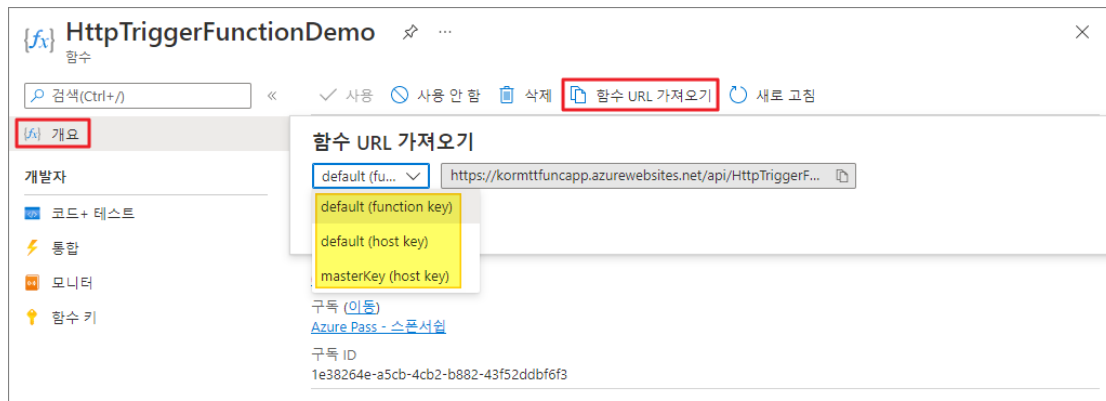
새 함수 *

HttpTriggerFunctionDemo

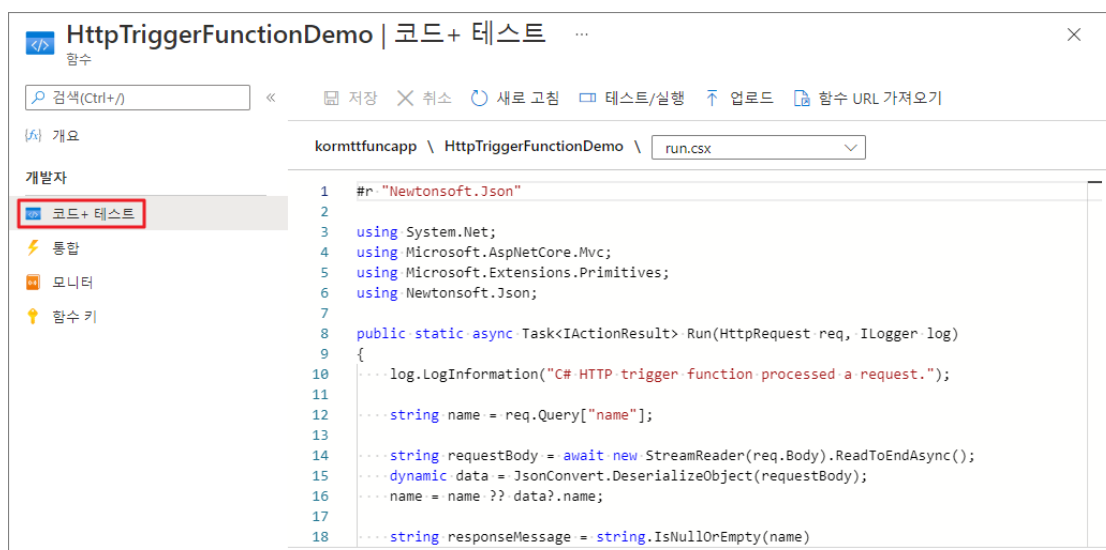
Authorization level * ⓘ

Function

12. 새로 만든 `[HttpTriggerFunctionDemo]` 함수 블레이드가 표시됩니다. `[HttpTriggerFunctionDemo]` 블레이드의 [개요]에서 새로 만든 Function의 기본 정보를 확인할 수 있습니다. Function URL을 가져오기 위해서는 [함수 URL 가져오기]를 클릭합니다. 그러면 `function key`, `host key`를 선택하여 원하는 Function URL을 가져올 수 있습니다.



13. [개발자 - 코드 + 테스트]로 이동하면 아래와 같이 기본 템플릿의 Function 코드가 표시됩니다. 메뉴의 [테스트/실행]을 클릭하면 브라우저에서 Function을 빠르게 테스트할 수 있습니다.



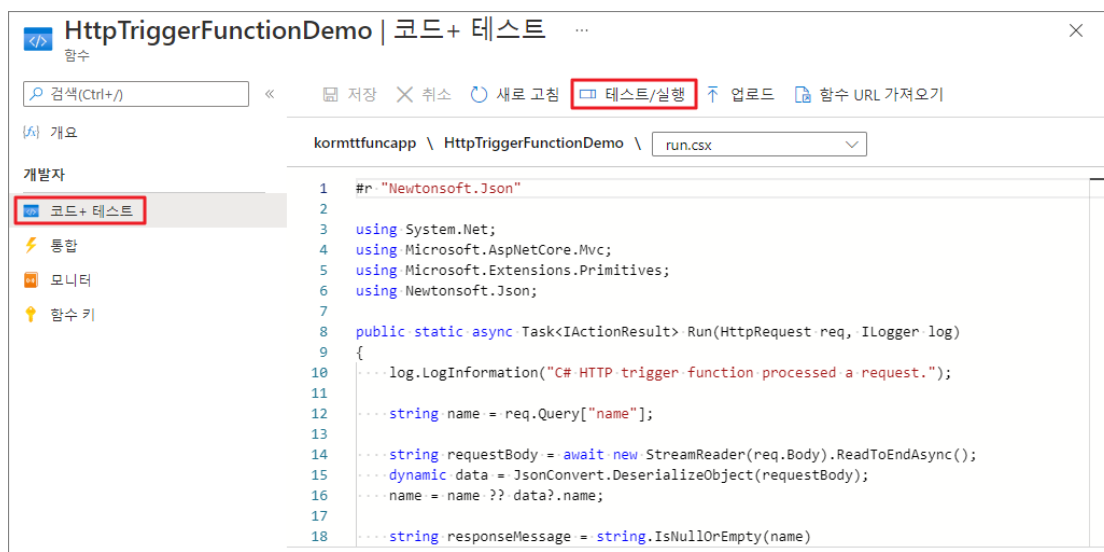
14. [HttpTriggerFunctionDemo 함수] 블레이드의 [개발자 - 통합]으로 이동합니다. 이 페이지에서 시각적으로 바인딩(binding)을 추가할 수 있습니다. 템플릿에서 선택한 트리거가 있으며 추가할 수 있는 여러 입력과 출력이 시각적으로 표시됩니다. 예를 들어 [입력 추가]를 클릭하면 Azure Blob Storage, Cosmos DB, Table Storage 등과 같이 사용할 수 있는 표준 입력 목록이 표시됩니다.



TASK 02. Azure 포털에서 코딩

빠른 프로토타입의 Function을 코딩하기 위해 Visual Studio나 Visual Studio Code를 사용하지 않고 Azure 포털을 사용할 수 있습니다. 이 작업에서는 Azure 포털을 사용하여 간단한 Function Apps를 만듭니다.

1. 앞서 만든 `HttpTriggerFunctionDemo` 함수] 블레이드의 [개발자 - 코드 + 테스트]로 이동한 후 코드 내용을 검토하고 메뉴에서 [테스트/실행]을 클릭합니다.
 - 이 코드는 `name`이라는 URL 매개 변수를 가져옵니다.
 - 그런 다음 구문 분석(parsing)을 수행하여 URL에 `name`이 없으면 쿼리 문자열이나 요청 본문에 `name`을 입력해야 한다는 내용을 출력하고 URL에 `name` 값이 있으면 "Hello, {name}. This HTTP..."을 출력합니다.



```
1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11     string name = req.Query["name"];
12
13     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
14     dynamic data = JsonConvert.DeserializeObject(requestBody);
15     name = name ?? data?.name;
16
17     string responseMessage = string.IsNullOrEmpty(name)
```

2. [입력] 탭에서 본문에 `{"name": "Azure"}` 내용이 입력되어 있는 것을 확인하고 [실행]을 클릭합니다.

입력 출력

HTTP 요청을 테스트하는 매개 변수를 제공합니다. 결과는 [출력] 탭에서 확인할 수 있습니다.

HTTP 메서드 ⓘ

POST

키

master (Host key)

쿼리

+ 매개 변수 추가

헤더

+ 헤더 추가

본문

```
1 {
2   "name": "Azure"
3 }
```

실행 닫기

3. 테스트 실행 결과가 [출력] 탭에 아래와 같이 표시되는 것을 확인합니다.

입력 출력

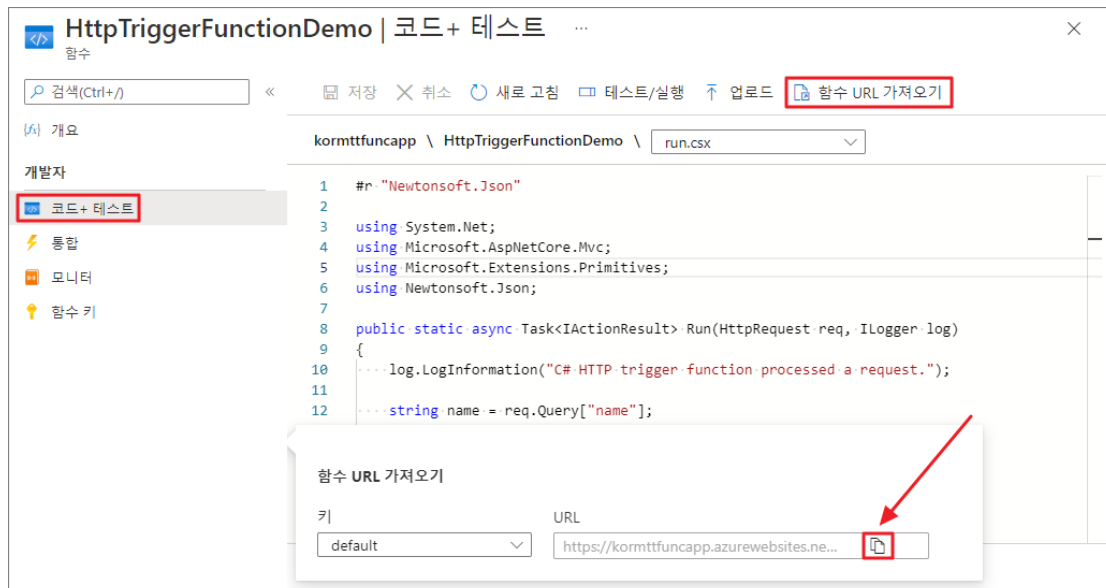
HTTP 응답 코드

200 OK

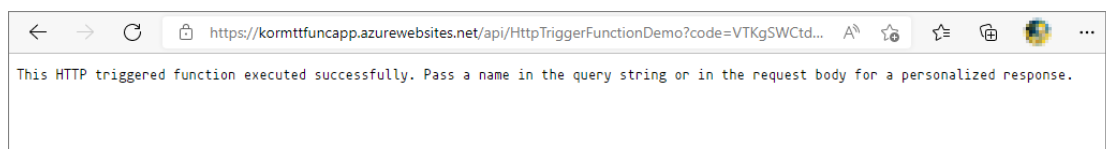
HTTP 응답 콘텐츠

Hello, Azure. This HTTP triggered function executed successfully.

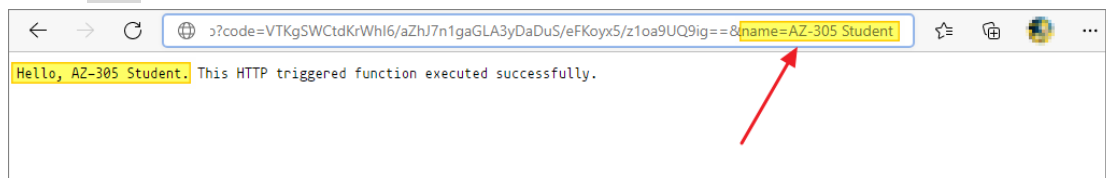
4. [HttpTriggerFunctionDemo 함수] 블레이드의 메뉴에서 [함수 URL 가져오기]를 클릭합니다. 함수 URL을 클립보드에 복사합니다.



5. 브라우저에서 새 탭을 열고 복사한 URL을 붙여 넣습니다. **name**에 아무런 값을 입력하지 않았기 때문에 아래와 같은 내용이 출력되는 것을 확인할 수 있습니다.



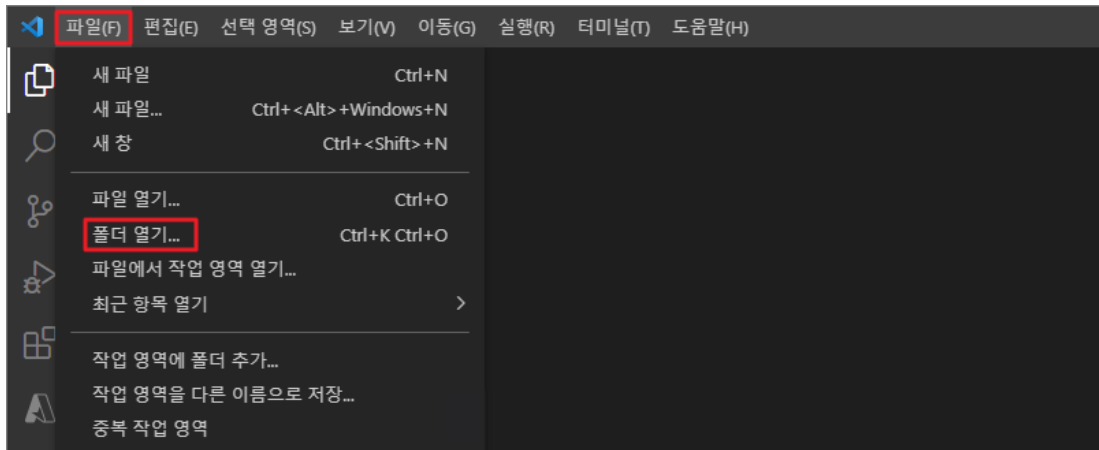
6. 브라우저의 URL 입력창에서 **&name=AZ-305 Student**를 입력하고 ENTER 키를 누르면 아래와 같이 입력한 **name** 값과 함께 내용이 출력되는 것을 확인할 수 있습니다.



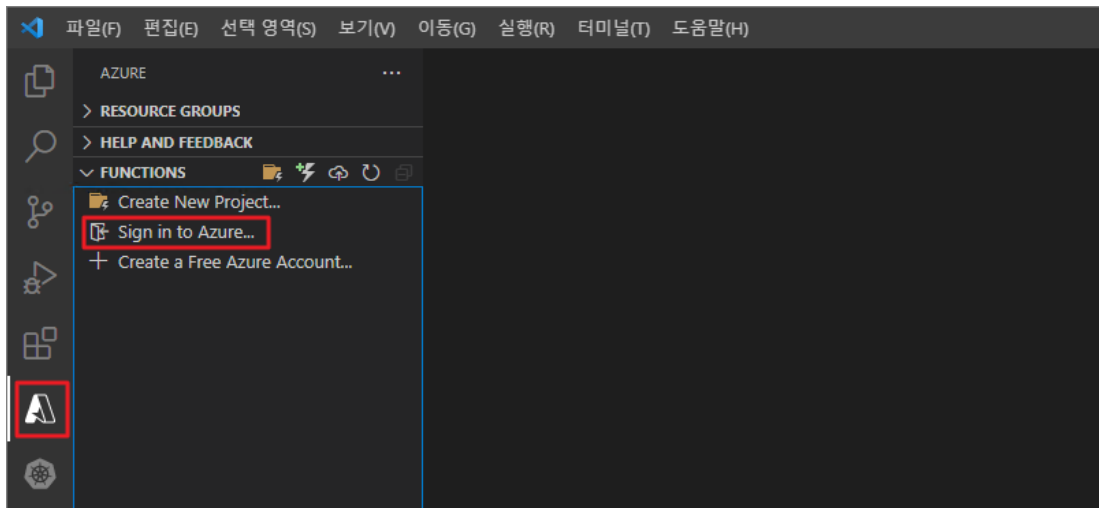
TASK 03. Visual Studio Code에서 개발

이 작업에서는 Visual Studio Code를 사용하여 Functions Apps를 개발합니다.

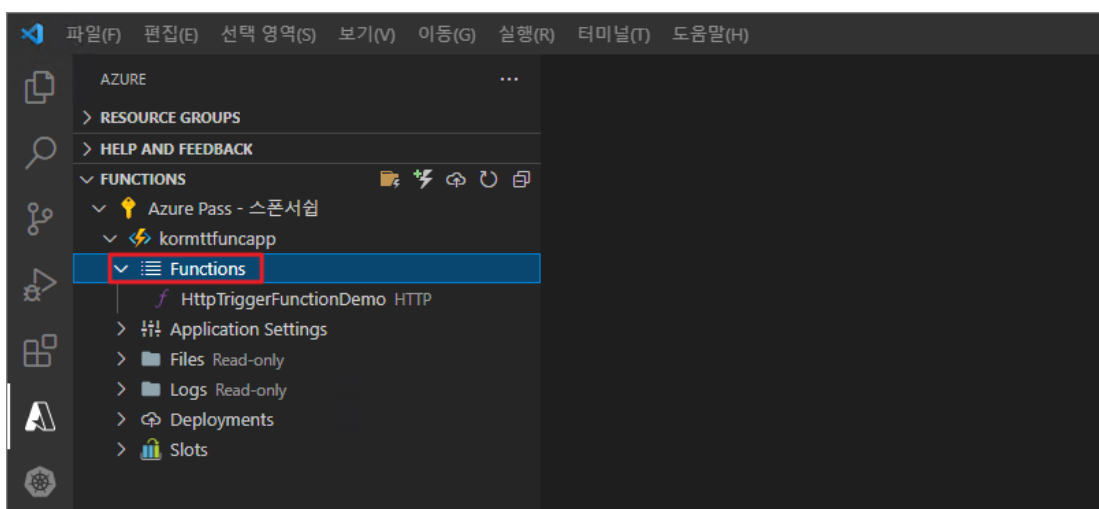
1. Visual Studio Code의 메뉴에서 [파일 - 폴더 열기]를 클릭합니다. **C:\02_functionApps** 폴더를 만들고 이 폴더를 선택합니다. [이 폴더에 있는 파일의 작성자를 신뢰합니까?] 창에서 [예, 작성자를 신뢰합니다]를 클릭합니다.



2. Visual Studio Code의 좌측 메뉴에서 [Azure]를 클릭합니다. [FUNCTIONS] 아래의 [Sign in to Azure]를 클릭하고 Azure 구독에 로그인합니다.

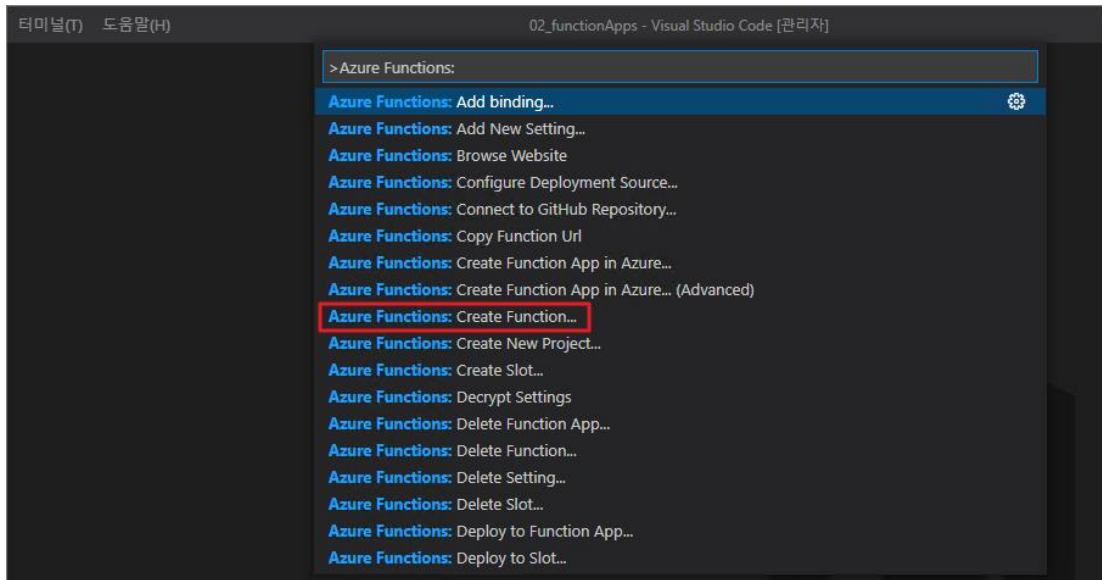


3. Azure 구독에 로그인하면 아래와 같이 Azure에 만들었던 Functions Apps 내용이 표시되는 것을 확인할 수 있습니다.

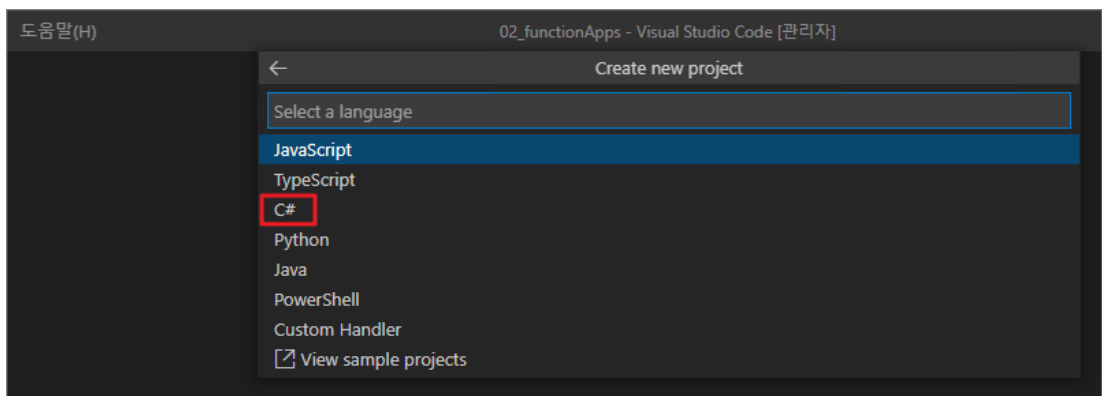


4. Visual Studio Code의 메뉴에서 [보기 - 명령 팔레트]를 클릭하거나 `Ctrl + <Shift> + P` 키를 누릅니다. 명령 팔레트에서 "Azure Functions"를 입력한 후 "Azure Functions: Create Function"을

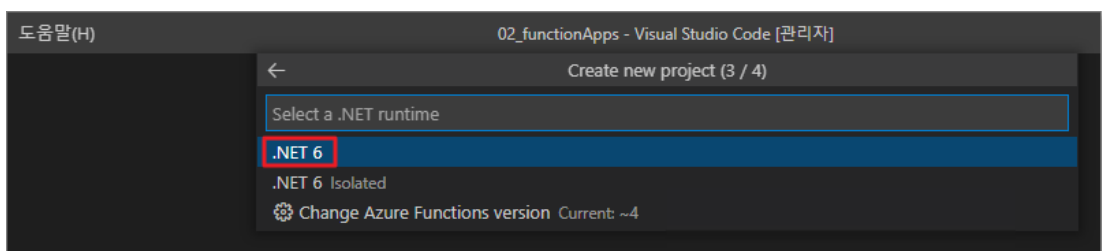
클릭합니다. `C:\02_functionApps` 폴더를 선택합니다.



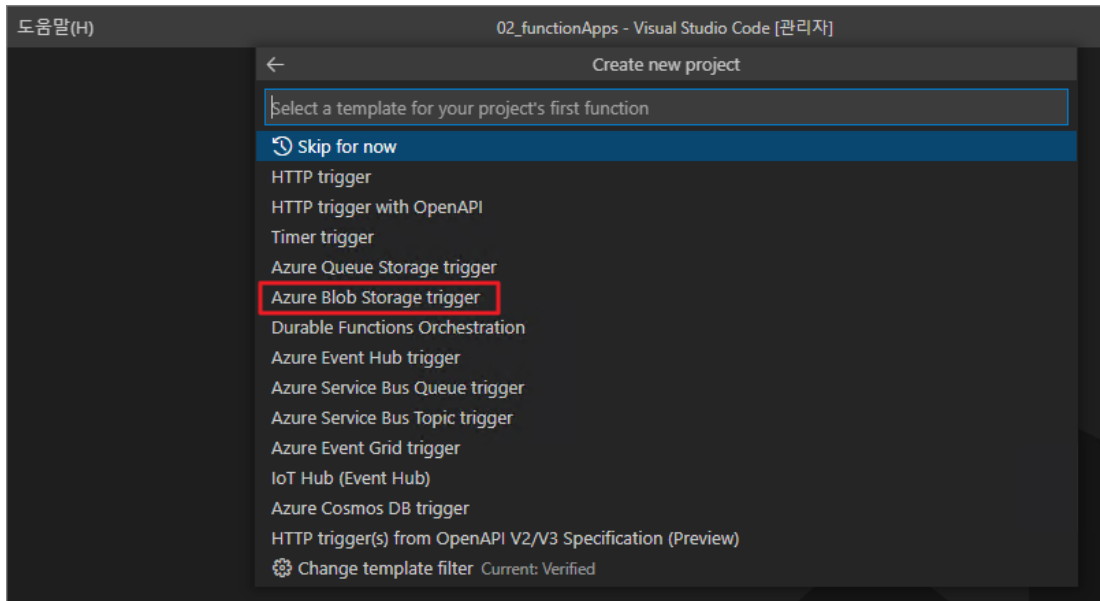
5. "Select a language"에서 "C#"을 선택합니다.



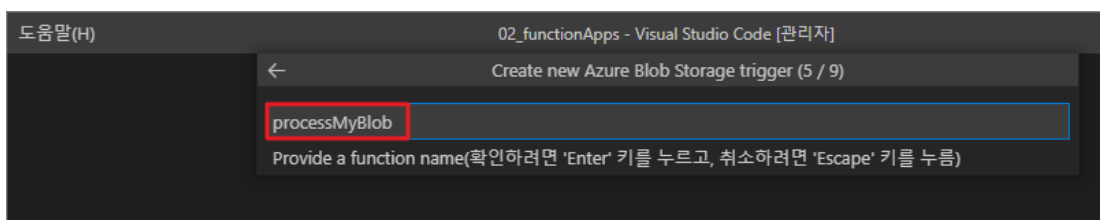
6. "Select a .NET runtime"에서 ".NET 6"을 선택합니다.



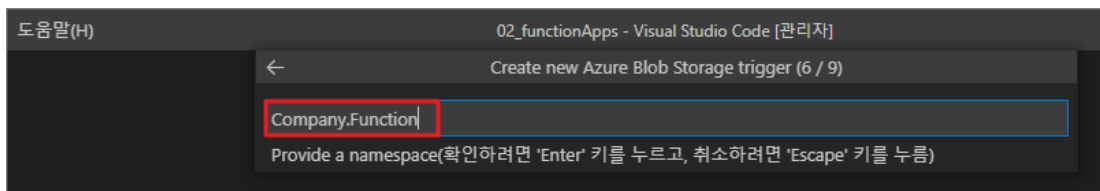
7. "Select a template for your project's first function"에서 "Azure Blob Storage trigger"를 클릭합니다. 이 실습에서는 Blob을 처리한 다음 다시 스토리지로 출력하는 Function을 만들 것입니다.



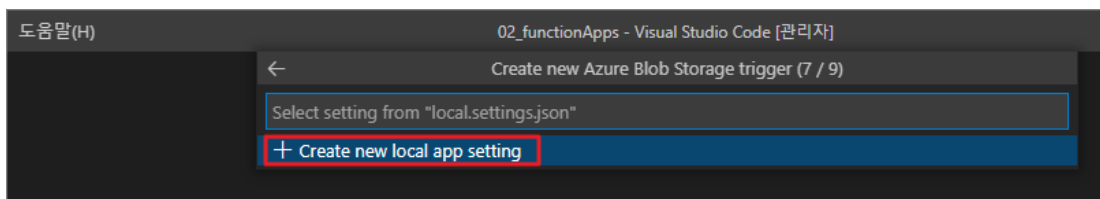
8. "Create new Azure Blob Storage trigger"에서 Function 이름에 "processMyBlob"을 입력하고 ENTER 키를 누릅니다.



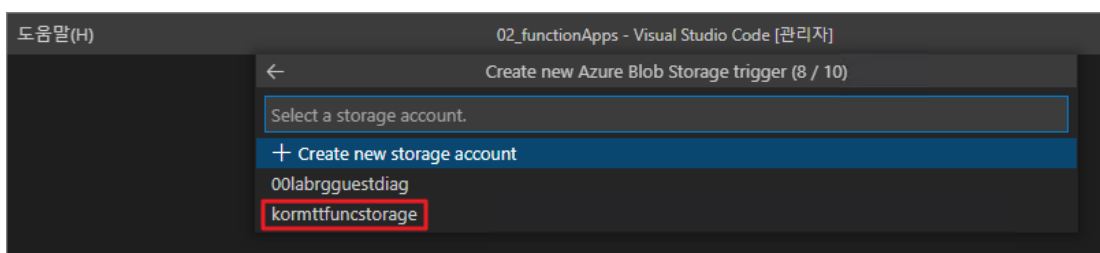
9. 네임스페이스 입력 화면에서 기본값을 유지하고 ENTER 키를 누릅니다.



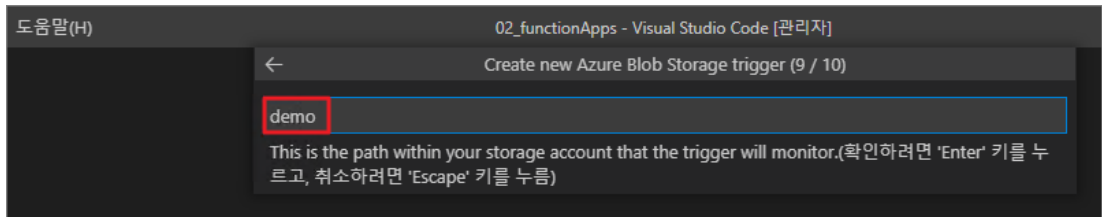
10. "local.settings.json" 파일 선택에서 [Create new local app setting]을 클릭합니다.



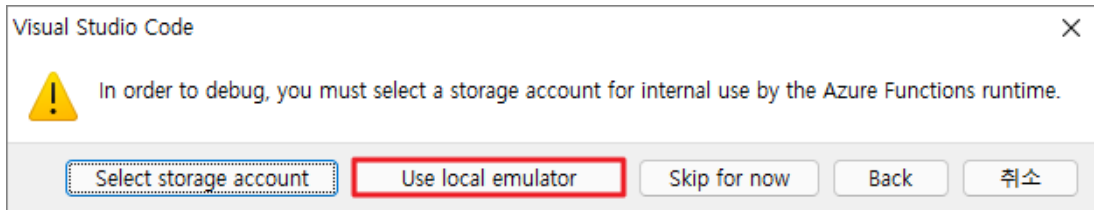
11. 스토리지 계정 선택 화면에서 실습을 위해 만들었던 스토리지 계정을 선택합니다.



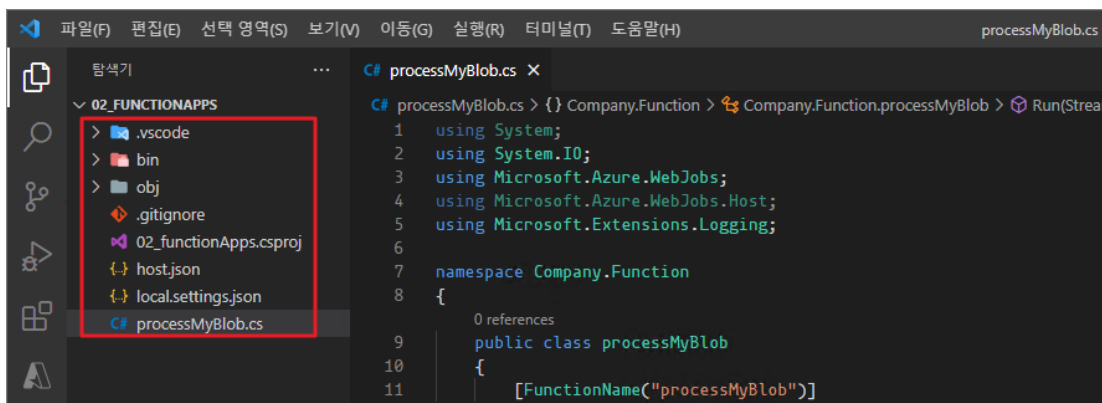
12. 스토리지 계정의 컨테이너 이름 입력에서 "demo"를 입력한 후 ENTER 키를 누릅니다.



13. 디버거를 위해 사용할 스토리지 계정을 묻는 창에서 [Use local emulator]를 클릭합니다.



14. 아래와 같이 Visual Studio Code에 Function 프로젝트가 생성된 것을 확인할 수 있습니다.



15. processMyBlob.cs 파일을 열고 다음과 같은 내용을 검토합니다.

- BlobTrigger를 통해 blob에 새 파일이 생길 때마다 이 Function이 실행됩니다.
- 처리되는 파일은 Connection 설정에서 제공되는 스토리지의 "demo" 컨테이너에 있어야 하며 파일 이름은 name 매개 변수에 전달됩니다.
- 이 실습에서는 앞서 Azure Functions를 배포할 때 지정한 스토리지 계정을 그대로 사용할 것이기 때문에 별도의 추가 스토리지가 필요하지 않습니다.

```
using System;
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;
namespace Company.Function
{
    public class processMyBlob
    {
        [FunctionName("processMyBlob")]
        public void Run(
            [BlobTrigger("demo/{name}", Connection = "YOUR_STORAGE_ACCOUNT")]Stream
            myBlob,
```

```

        string name, ILogger log)
    {
        log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
    }
}
}

```

processMyBlob.cs X

```

processMyBlob.cs > ...
1  using System;
2  using System.IO;
3  using Microsoft.Azure.WebJobs;
4  using Microsoft.Azure.WebJobs.Host;
5  using Microsoft.Extensions.Logging;
6
7  namespace Company.Function
8  {
9      0 references
10     public class processMyBlob
11     {
12         [FunctionName("processMyBlob")]
13         0 references
14         public void Run(
15             [BlobTrigger("demo/{name}", Connection = "kormttfuncstorage_STORAGE")] Stream myBlob,
16             string name, ILogger log)
17         {
18             log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
19         }
20     }

```

16. Visual Studio Code에서 `local.settings.json` 파일을 클릭합니다. 아래와 같이 Function Apps의 [애플리케이션 설정]에 구성되어 있는 값이 표시됩니다.

- 이 작업에서는 blob에 연결하는 방법으로 애플리케이션 설정에 지정되어 있는 `AzureWebJobsStorage`라는 연결을 사용할 것입니다.
- 따라서 로컬 개발에서는 로컬 스토리지를 의미하는(`UseDevelopmentStorage=true`) 이 연결을 사용하며 Azure 포털에서는 애플리케이션 설정의 `AzureWebJobsStorage` 지정되어 있는 스토리지를 사용합니다.
- `ProcessMyBlob.cs` 파일의 `Connection`에 정의되어 있던 연결은 사용하지 않을 것이기 때문에 삭제합니다.

```

{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
}

```

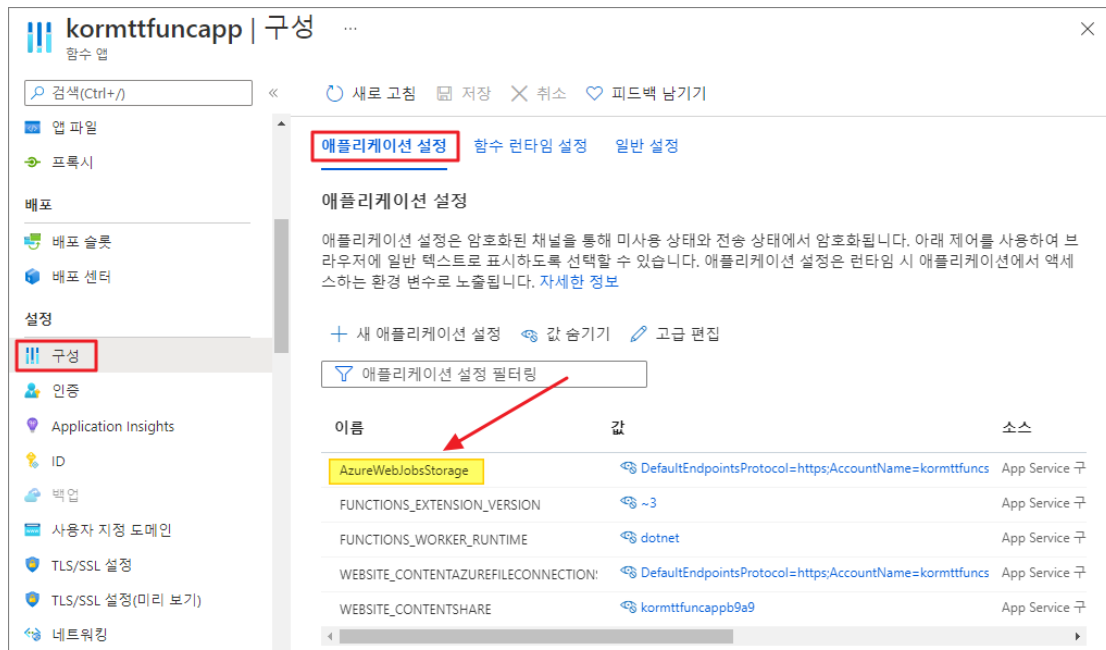
local.settings.json X

```

local.settings.json > ...
1  {
2    "IsEncrypted": false,
3    "Values": {
4      "AzureWebJobsStorage": "UseDevelopmentStorage=true",
5      "FUNCTIONS_WORKER_RUNTIME": "dotnet",
6      "kormttfuncstorage_STORAGE": "DefaultEndpointsProtocol=https;AccountName=kormttfuncstorage;AccountKey=
7    }
8  }
9

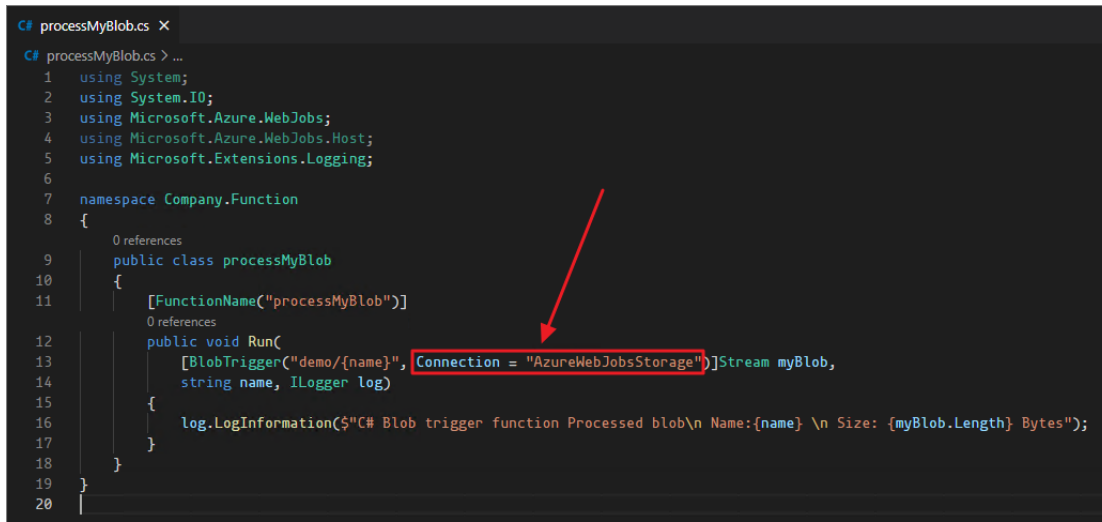
```

삭제



17. processMyBlob.cs 파일로 이동한 후 Connection의 값을 위에서 애플리케이션 설정의 AzureWebJobsStorage로 변경합니다.

```
using System;
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;
namespace Company.Function
{
    public class processMyBlob
    {
        [FunctionName("processMyBlob")]
        public void Run(
            [BlobTrigger("demo/{name}", Connection = "AzureWebJobsStorage")] Stream
myBlob,
            string name, ILogger log)
        {
            log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n
Size: {myBlob.Length} Bytes");
        }
    }
}
```

```

1  using System;
2  using System.IO;
3  using Microsoft.Azure.WebJobs;
4  using Microsoft.Azure.WebJobs.Host;
5  using Microsoft.Extensions.Logging;
6
7  namespace Company.Function
8  {
9      0 references
10     public class processMyBlob
11     {
12         [FunctionName("processMyBlob")]
13         0 references
14         public void Run(
15             [BlobTrigger("demo/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob,
16             string name, ILogger log)
17         {
18             log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
19         }
20     }

```

18. porcessMyBlob.cs 파일에서 다음과 같은 내용을 수정합니다.

- ① `log.LogInformation`: Function에서 처리하는 blob의 이름과 byte 길이를 표시하는 내용이므로 삭제하고 이 내용을 `var len = myBlob.Length`로 변경합니다.
- ② blob에서 다시 출력하기 위해 `BlobTrigger`와 유사한 Blob을 사용하고 blob에 쓰기 작업을 위해 `FileAccess.Write`를 지정합니다. 출력 변수는 `outputBlob`으로 지정합니다.
- ③ `myBlob`을 `CopyTo`를 사용하여 `outputBlob`에 복사하는 코드를 추가합니다.

```

using System;
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;
namespace Company.Function
{
    public class processMyBlob
    {
        [FunctionName("processMyBlob")]
        public void Run(
            [BlobTrigger("demo/{name}", Connection = "AzureWebJobsStorage")]Stream
myBlob,
            [Blob("output/{name}", FileAccess.Write, Connection =
"AzureWebJobsStorage")]Stream outputBlob,
            string name, ILogger log)
        {
            var len = myBlob.Length;

            myBlob.CopyTo(outputBlob);
        }
    }
}

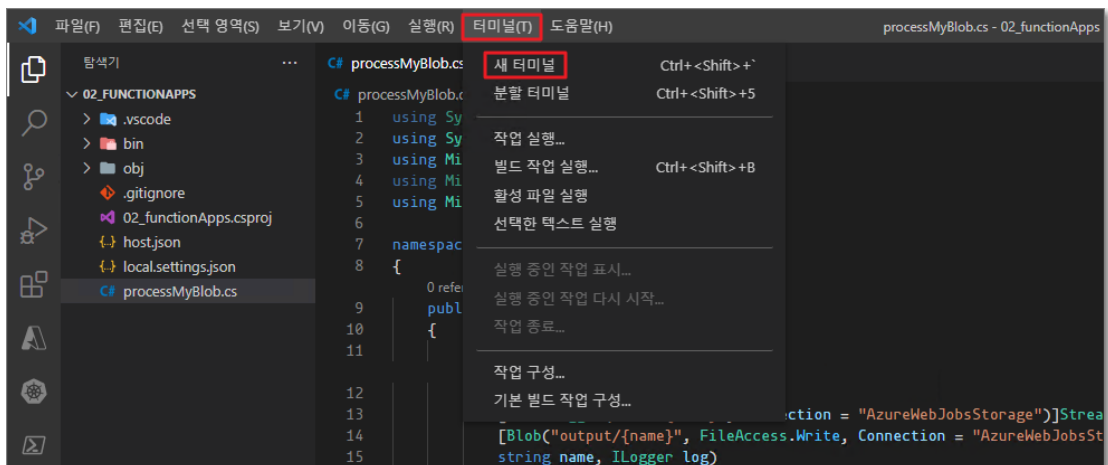
```

```

C# processMyBlob.cs X
C# processMyBlob.cs > ...
1  using System;
2  using System.IO;
3  using Microsoft.Azure.WebJobs;
4  using Microsoft.Azure.WebJobs.Host;
5  using Microsoft.Extensions.Logging;
6
7  namespace Company.Function
8  {
9      0 references
10     public class processMyBlob
11     {
12         [FunctionName("processMyBlob")]
13         0 references
14         public void Run(
15             [BlobTrigger("demo/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob,
16             [Blob("output/{name}", FileAccess.Write, Connection = "AzureWebJobsStorage")]Stream outputBlob,
17             string name, ILogger log)
18         {
19             var len = myBlob.Length;
20             myBlob.CopyTo(outputBlob);
21         }
22     }
23 }

```

19. 이제 작성한 Function Apps를 실행할 수 있습니다. Function을 로컬에서 실행하거나 Azure에 빠르게 배포할 수 있습니다. Visual Studio Code의 메뉴에서 [터미널 - 새 터미널]을 클릭합니다.



20. PowerShell 터미널 창에서 작성한 Function을 빌드하기 위해 다음 명령을 실행합니다.

```
# 애플리케이션 빌드
dotnet build
```

```

문제  출력  디버그 콘솔  터미널

PowerShell 7.2.2
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\02_functionApps> # 애플리케이션 빌드
PS C:\02_functionApps> dotnet build
.NET용 Microsoft (R) Build Engine 버전 17.1.0+ae57d105c
Copyright (C) Microsoft Corporation. All rights reserved.

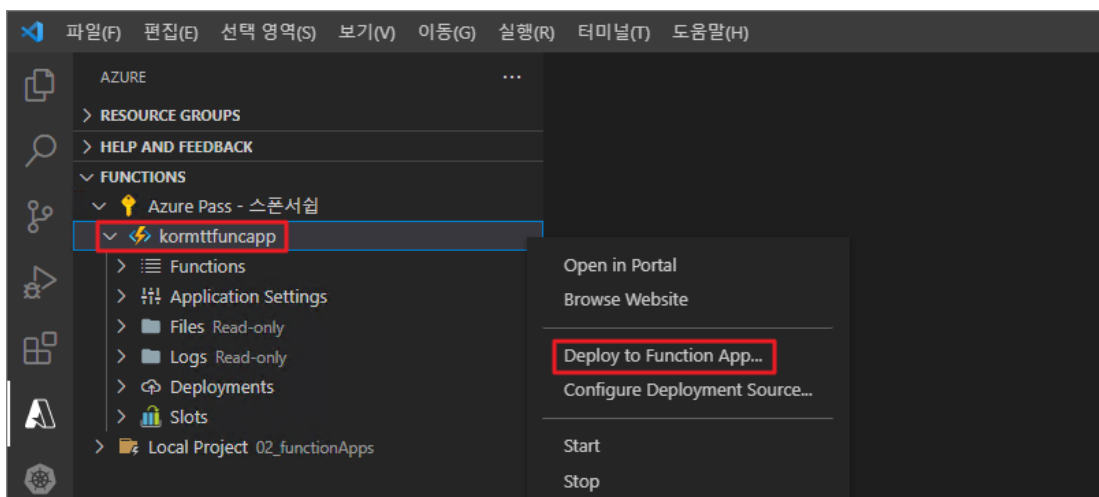
복원할 프로젝트를 확인하는 중...
C:\02_functionApps\02_functionApps.csproj을(를) 1.29 sec 동안 복원했습니다.
02_functionApps -> C:\02_functionApps\bin\Debug\net6.0\02_functionApps.dll

빌드했습니다.
경고 0개
오류 0개

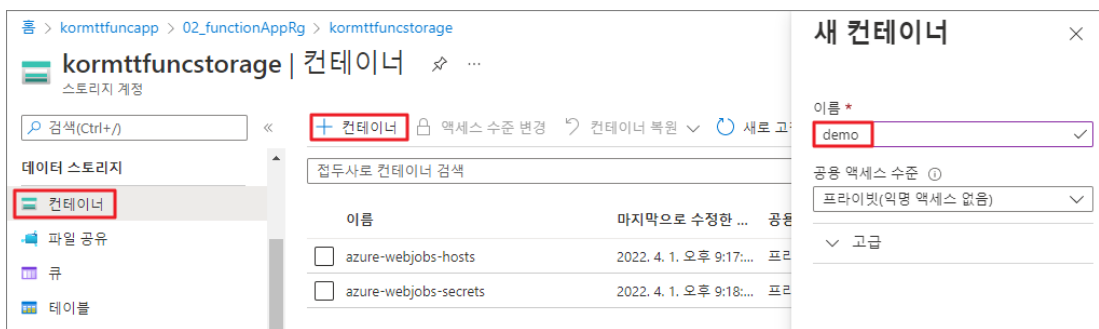
경과 시간: 00:00:25.15
PS C:\02_functionApps>

```

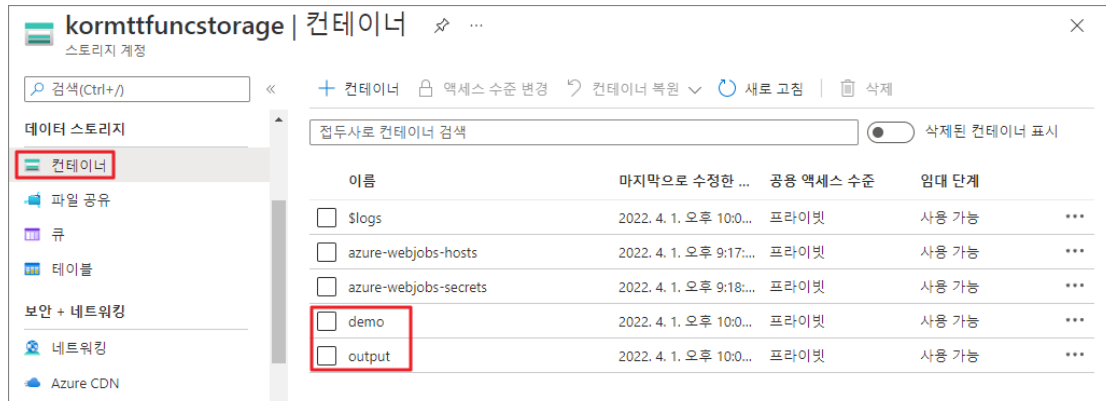
21. Visual Studio Code의 좌측 메뉴에서 [Azure]를 클릭합니다. [FUNCTIONS - Function Apps 이름]을 마우스 우 클릭하고 [Deploy to Function App]을 클릭합니다. 기존 배포를 덮어쓸 것인지 묻는 창에서 [Deploy]를 클릭합니다. Function App 런타임 관련 메시지가 표시되면 [Deploy Anyway]를 클릭합니다.



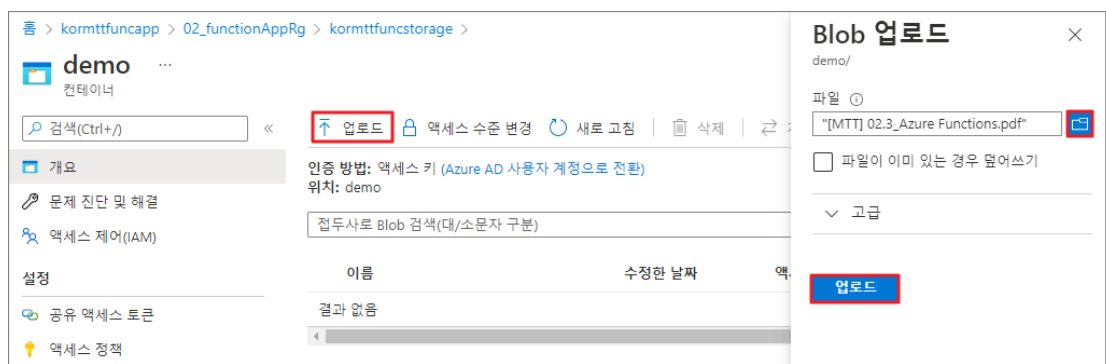
22. Visual Studio Code 터미널 창에서 배포가 완료되면 Azure 포털로 이동합니다. [스토리지 계정] 블레이드의 [데이터 스토리지 - 컨테이너]로 이동한 후 메뉴에서 [컨테이너]를 클릭합니다. [새 컨테이너]에서 이름에 "demo"를 입력하고 [만들기]를 클릭합니다.



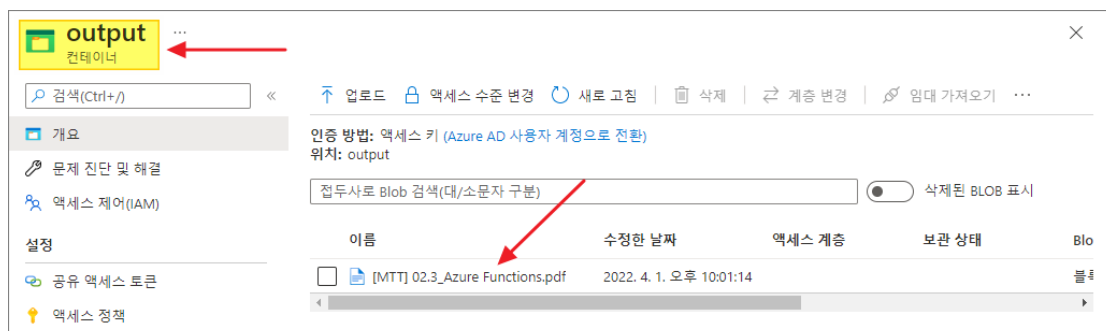
23. 동일한 방법으로 "output" 이름의 컨테이너를 하나 더 만듭니다.



24. [demo 컨테이너] 블레이드로 이동한 후 메뉴에서 [업로드]를 클릭합니다. [Blob 업로드]에서 임의의 파일을 선택한 후 [업로드]를 클릭합니다.

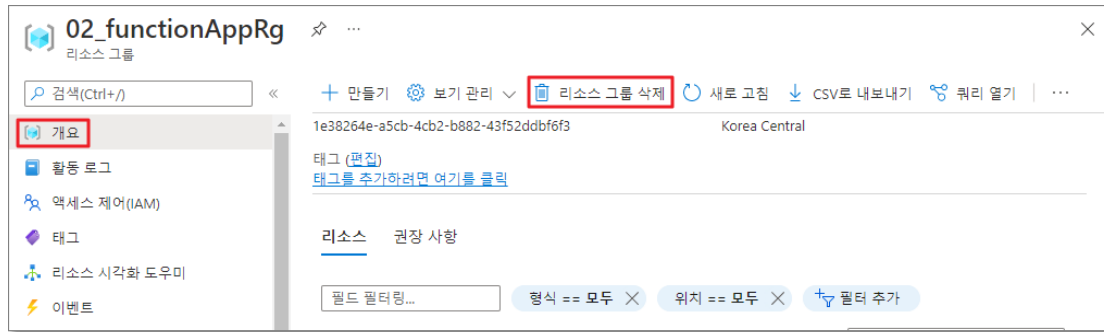


25. [output 컨테이너] 블레이드로 이동하면 Function Apps에 의해 복사된 파일이 표시되는 것을 확인할 수 있습니다.



TASK 04. 리소스 정리

1. Azure 포털에서 [02_functionAppRg 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.

