**Azure Functions**

**HOL 가이드**

|  |  |  |
| --- | --- | --- |
|  | **문서명** | Azure Functions HOL 가이드 |
|  | **버전** | 1.2 |
|  |  |  |

# 문서 이력 관리

|  |  |  |  |
| --- | --- | --- | --- |
| **문서 버전** | **작성일** | **작성자** | **변경 사항** |
| 1.0 | 2024-03-28 | 김민서 | 초안 작성 |
| 1.1 | 2024-04-17 | 김민서 | 오타 수정  HttpTrigger 추가코드 삭제  Python QueueTrigger 스토리지 계정 생성 순서 변경  목차 페이지 수정 |
| 1.2 | 2024-04-18 | 김민서 | queue 데이터 인입 확인을 위한 http트리거 추가 |
| 1.3 | 2024-04-18 | 김민서 | 오타 수정 |
|  |  |  |  |
|  |  |  |  |

**목차**

[문서 이력 관리 2](#_Toc141794075)

[Lab 0 – HOL 환경 설명 4](#_Toc141794076)

[Task 1. Hands-on Lab 목표 4](#_Toc141794077)

[Task 2. 사전 준비사항 4](#_Toc141794078)

[Lab 1 – VSCode 로컬환경 구성 5](#_Toc141794080)

[Task 1. VSCode 로컬환경 구성 5](#_Toc141794081)

[Lab 2 – Java Function 생성 및 실행 6](#_Toc141794083)

[Task 1. HttpTrigger 생성 6](#_Toc141794084)

[Task 2. HttpTrigger 로컬환경 실행 9](#_Toc141794085)

[Task 3. BlobTrigger 추가 10](#_Toc141794085)

[Task 4. 스토리지 계정 리소스 생성 1](#_Toc141794085)1

[Task 5. BlobTrigger 로컬환경 실행 15](#_Toc141794085)

[Lab 3 –Java Function 배포 19](#_Toc141794086)

[Task 1. HttpTrigger, BlobTrigger 코드 수정 1](#_Toc141794085)9

[Task 2. BlobTrigger 이미지 리사이징 테스트](#_Toc141794085) 22

[Task 3. HttpTrigger, BlobTrigger 배포 23](#_Toc141794085)

[Task 4. Azure Portal BlobTrigger 테스트 27](#_Toc141794085)

[Lab 4 – Python funcion 생성 및 실행 29](#_Toc141794090)

[Task 1. MySQL Database 생성 29](#_Toc141794092)

[Task 2. QueueTrigger 생성 33](#_Toc141794091)

[Task 3. QueueTrigger 로컬환경 실행 36](#_Toc141794092)

[Task 4. 데이터 인입 확인을 위한 http트리거 추가 39](#_Toc141794092)

[Task 5. 코드 수정 및 테스트 4](#_Toc141794092)1

[Task 5. QueueTrigger 테스트 4](#_Toc141794092)4

[Lab 5 – Python funcion 배포 4](#_Toc141794090)6

[Task 1. Azure Portal Function 생성 4](#_Toc141794091)6

[Task 2. Python QueueTrigger 배포 및 데이터 확인 47](#_Toc141794091)

# Lab 0 – HOL 환경 설명

## Hands-on Lab 목표

1. VSCode를 사용하여 Azure Functions의 로컬 환경을 구성합니다.
2. Azure Functions코드를 작성하고 Azure Portal에 배포합니다.
3. Azure Portal에 배포된 서비스를 확인합니다.

## 사전 준비

1. Azure Portal 접근 가능 구독
2. VSCode Tool
3. VSCode 내 Azure Extension 설치
4. JDK 17, maven 설치 필요
5. Python 3.10
6. Postman
7. 테스트용 텍스트파일
8. DB Tool

# Lab 1 – VSCode 로컬 환경 구성

## VSCode With Azure 연동

### VSCode 실행 후 좌측 상단 RESOURCES에서 “Sign in to Azure…”를 선택합니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

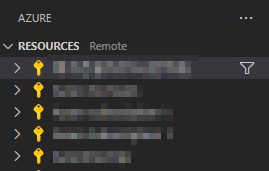
자동 생성된 설명

### 연동할 계정을 선택해서 VSCode와 연동 작업을 진행합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

### 아래 이미지와 같이 구독 리스트를 확인합니다.



# Lab 2 – Functions 생성 및 실행

* HTTP 트리거는 외부에서 HTTP요청시 함수를 트리거하는 메커니즘으로 동작합니다. 주요 특징으로 HTTP 메서드를 지원하기 때문에 다양한 유형의 요청에 대응이 가능하며 개발자 입장에서 서버 관리에 대한 부담 없이 신속하게 확장 가능한 웹 애플리케이션을 구축할 수 있습니다.

## httpTrigger 생성

1. WORKSPACE에 있는 Create Function 아이콘을 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 작업할 위치 디렉토리를 선택합니다

텍스트, 폰트, 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. Java를 선택합니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 설치된 JDK버전에 따라 Java 버전을 선택합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 사용할 그룹 아이디를 지정합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 사용할 Functions의 id를 입력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 스냅샷 파일명을 지정합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Java 패키지명을 지정합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Functions에서 사용할 App Name을 지정합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Maven을 선합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

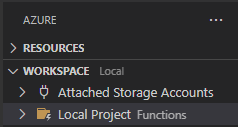
자동 생성된 설명

1. Maven 빌드가 완료될 때까지 기다립니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

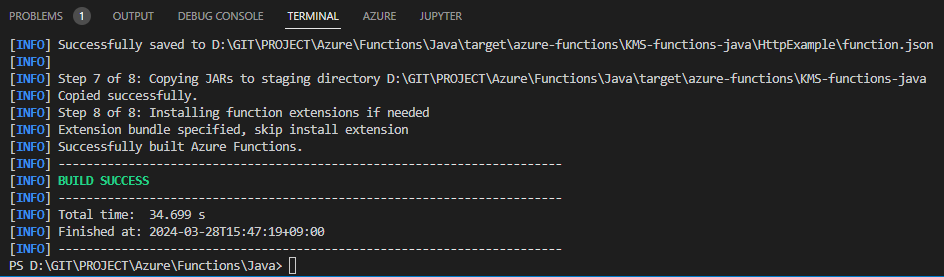
1. 아래 이미지와 같이 Local Project가 생성되면 성공입니다.



## httpTrigger 로컬환경 실행

1. 터미널에서 다음과 같은 명령어를 입력해 프로젝트를 maven명령어로 build작업을 진행합니다.

|  |
| --- |
| mvn clean package |



1. 다음 명령어를 입력해 로컬에서 function을 동작 시킵니다.

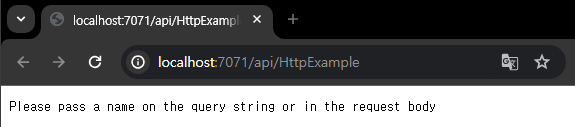
|  |
| --- |
| mvn azure-functions:run |

1. 명령어 입력 후 제공되는 로컬url을 복사하여 간단한 테스트를 진행합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 크롬 브라우저를 열어 복사한 url을 입력 후 get방식으로 파라미터를 전달합니다.
   * http://localhost:7071/api/HttpExample?name=java



1. 아래 이미지와 같이 Hello, java가 확인되면 성공입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

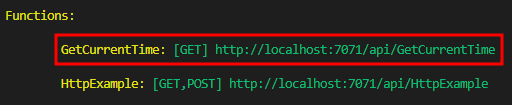
1. http 트리거에 현재 시간을 알려주는 코드를 추가합니다.

|  |
| --- |
| import java.text.SimpleDateFormat;  import java.util.Calendar; |
| @FunctionName("GetCurrentTime")      public HttpResponseMessage run2(              @HttpTrigger(name = "req", methods = {HttpMethod.GET},                          authLevel = AuthorizationLevel.ANONYMOUS)                          HttpRequestMessage<Optional<String>> request,              final ExecutionContext context) {          context.getLogger().info("Java HTTP trigger processed a request.");          // 현재 시간을 가져오기 위해 Calendar 객체 사용          Calendar cal = Calendar.getInstance();          SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");          String currentTime = sdf.format(cal.getTime());          return request.createResponseBuilder(HttpStatus.OK).body("Current time is: " + currentTime).build();      } |

1. 터미널에서 ctrl+c를 눌러 기존 작업 종료 후 maven명령어를 이용해 build작업 후 function을 실행합니다.

|  |
| --- |
| mvn clean package |
| mvn azure-functions:run |

1. 새로 추가되는 url을 클릭해서 브라우저를 확인합니다.



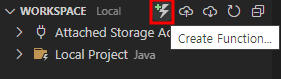
1. 해당 url에 접속하여 현재시간을 확인합니다. [완료]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

## Java Functions BlobTrigger 추가

1. Create Function… 버튼을 클릭해서 Function 생성작업을 진행합니다.

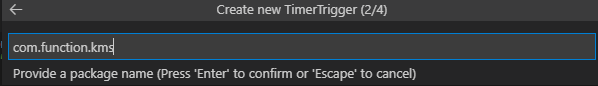


1. BlobTrigger를 선택합니다.

텍스트, 폰트, 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 패키지 경로를 입력합니다.



1. BlobTrigger의 이름(BlobTriggerJava-OUTPUT)을 지정합니다.

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

1. 스토리지 계정과 연동할 환경변수를 선택합니다. (AzureWebJobsStorage)

텍스트, 폰트, 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 사용할 컨테이너명을 입력합니다. (output)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. BlobTrigger 추가가 완료되었습니다. [완료]

## 스토리지 계정 생성

1. 스토리지 계정 생성을 위해 Azure Portal로 접속 후 스토리지 계정을 검색 후 클릭하여 생성을 진행합니다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

1. 만들기를 클릭합니다.

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

1. “**기본 사항**” 탭에서 리소스 그룹에 새로 만들기를 클릭해서 리소스 그룹과 스토리지 계정 이름을 입력 후 “다음”을 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. “**고급**” 탭에서 “개별 컨테이너에 대한 익명 액세스 허용”에 체크를 한 다음 “검토+만들기”를 클릭합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. “검토 + 만들기” 탭에서 “만들기”를 클릭하여 스토리지 계정 생성을 마무리합니다.

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 배포가 완료될 때까지 기다립니다.

텍스트, 폰트, 번호, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 배포가 완료되었습니다. “리소스로 이동”을 클릭해서 개요로 이동합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 컨테이너를 생성하기 위해 데이터 스토리지에 있는 “컨테이너”를 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 컨테이너를 생성하기 위해 “+ 컨테이너” 버튼을 클릭합니다.

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

1. 새 컨테이너에서 이름을 입력 후 “익명 액세스 수준”을 “컨테이너(컨테이너와 Blob에 대한 익명 읽기 액세스)”로 선택한 다음 “만들기”를 클릭합니다.
   * + input, output 두개의 컨테이너 생성

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 컨테이너 생성이 완료되었습니다. [완료]

텍스트, 라인, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

## Java Functions BlobTrigger 로컬 실행

1. Functions를 실행하기 전 스토리지 계정과 연동할 연결 문자열을 복사하기 위해 “보안 + 네트워킹”에 있는 “액세스 키”를 클릭합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. “연결 문자열”에 표시버튼을 클릭해서 연결 문자열을 복사합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. VSCode로 돌아와 local.settings.json파일을 열어 복사한 연결 문자열을 AzureWebJobsStorage에 붙여 넣은 후 저장합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 기존 Functions이 로컬에서 동작 중이라면 ctrl+c를 눌러 기존 작업을 종료한 후 mvn 명령어를 사용하여 build작업 후 function을 동작 시킵니다.

|  |
| --- |
| Build : mvn clean package |
| Function 실행 : mvn azure-functions:run |

* + - * **중요** : 만약 mvn azure-functions:run명령어 실행 후 아래와 같은 에러가 발생하는 경우, host.json파일을 아래와 같이 수정이 필요합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + host.json파일을 열어 아래와 같이 코드를 수정합니다.

|  |
| --- |
| {  "version": "2.0",  "logging": {  "applicationInsights": {  "samplingSettings": {  "isEnabled": true,  "excludedTypes": "Request"  }  }  },  "extensionBundle": {  "id": "Microsoft.Azure.Functions.ExtensionBundle",  "version": "[3.3.0, 3.9.0)"  }  } |

* + 다시 build 후 function 실행 명령어 수행

1. Functions이 실행된 후 아래와 같은 url을 확인합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Azure Portal로 돌아가 생성한 output컨테이너를 클릭합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 업로드를 클릭한 후 Blob 업로드에서 테스트용 이미지를 업로드 합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. Blob 업로드에서 이미지 업로드 후 업로드 버튼을 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 이미지 업로드 후 실행중인 function 로그상에서 해당 이미지에 대한 정보가 아래와 같이 나오면 성공입니다. [완료]

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

* + 추후 작업을 위해 ctrl+c를 눌러 작업을 종료시킵니다.

# Lab 3 – Function 배포

## httpTrigger, blobTrigger 코드 수정

* 아래 코드는 GitHub에서 제공됩니다. GitHub를 clone해도 무방합니다.
* <https://github.com/mtcdevops/Azure_Functions>

1. 기존 pom.xml에서 아래 의존성을 추가합니다.

|  |
| --- |
| <!-- 이미지 업로드 종속성 추가-->  <dependency>  <groupId>com.microsoft.azure.functions</groupId>  <artifactId>azure-functions-java-library</artifactId>  <version>3.1.0</version>  </dependency>  <dependency>  <groupId>com.microsoft.azure</groupId>  <artifactId>azure-storage</artifactId>  <version>8.6.5</version>  </dependency>  <!-- 썸네일 사용을 위한 의존성 추가 -->  <dependency>  <groupId>net.coobird</groupId>  <artifactId>thumbnailator</artifactId>  <version>0.4.8</version>  </dependency>  <!--BlobInputStream, BlobOutputStream 사용을 위한 의존성 추가-->  <dependency>  <groupId>com.azure</groupId>  <artifactId>azure-storage-blob</artifactId>  <version>12.25.3</version>  </dependency>  <!-- SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder". -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.13.5</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.13.5</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.dataformat</groupId>  <artifactId>jackson-dataformat-xml</artifactId>  <version>2.13.5</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.datatype</groupId>  <artifactId>jackson-datatype-jsr310</artifactId>  <version>2.13.5</version>  </dependency> |

1. 이미지 업로드시 파일정보를 알아내기 위해 blobTrigger를 수정합니다.
   * Import 추가

|  |
| --- |
| import java.io.IOException;  import java.io.ByteArrayInputStream;  import java.io.ByteArrayOutputStream;  import javax.imageio.ImageIO;  import java.awt.image.BufferedImage;  import com.azure.storage.blob.\*;  import com.microsoft.azure.functions.ExecutionContext;  import com.microsoft.azure.functions.annotation.FunctionName; |

* + BlobTrigger-OUTPUT 트리거 추가

|  |
| --- |
| @FunctionName("BlobTriggerJava-OUTPUT")  @StorageAccount("AzureWebJobsStorage")  public void run(  @BlobTrigger(name = "content", path = "**output**/{name}", dataType = "binary") byte[] content,  @BindingName("name") String name,  final ExecutionContext context  ) {    ByteArrayInputStream output = new ByteArrayInputStream(content);  try {  BufferedImage originalImage = ImageIO.read(output);  int originalWidth = originalImage.getWidth();  int originalHeight = originalImage.getHeight();    if (name.startsWith("resizedImage")) {// 이미지 중복 체크  context.getLogger().info("이미지가 output 컨테이너 Blob 저장소에 저장되었습니다. 파일명 :"+name);  context.getLogger().info("Image Size: " + originalWidth + "x" + originalHeight);  } else {  context.getLogger().info("Java Blob trigger function processed a blob. Name: " + name + "\n Size: " + content.length + " Bytes");  }  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  **}**  } |

1. 이미지 리사이징을 위한 BlobTrigger-INPUT 코드(run2 메서드)를 추가합니다.

|  |
| --- |
| /\* 저장대상 \*/  private final String blobContainer = "output";    @FunctionName("BlobTriggerJava-INOUT")  @StorageAccount("AzureWebJobsStorage")  public void run2(  @BlobTrigger(name = "content", path = "input/{name}", dataType = "binary") byte[] content,  @BindingName("name") String name,  final ExecutionContext context  ) {  // Load image from input stream  try {    // 1. Load the input image  ByteArrayInputStream input = new ByteArrayInputStream(content);  BufferedImage originalImage = ImageIO.read(input);    int 비율설정 = 2;  // Get the dimensions of the original image  int originalWidth = originalImage.getWidth();  int originalHeight = originalImage.getHeight();  int resizedWidth = originalImage.getWidth()/비율설정;  int resizedHeight = originalImage.getHeight()/비율설정;    // 2. Resize the image  BufferedImage resizedImage = new BufferedImage(resizedWidth, resizedHeight, originalImage.getType());  resizedImage.createGraphics().drawImage(originalImage, 0, 0, resizedWidth, resizedHeight, null);    // 3. 처리된 이미지를 Blob 저장소에 업로드  // 처리된 이미지를 ByteArrayOutputStream에 저장합니다.  ByteArrayOutputStream output = new ByteArrayOutputStream();  ImageIO.write(resizedImage, "png", output);    // Azure Blob Storage 클라이언트 초기화  // BlobServiceClient blobServiceClient = new BlobServiceClientBuilder().connectionString(storageConnectionString).buildClient();  BlobServiceClient blobServiceClient = new BlobServiceClientBuilder().connectionString(System.getenv("AzureWebJobsStorage")).buildClient();  BlobContainerClient containerClient = blobServiceClient.getBlobContainerClient(blobContainer);  containerClient.getBlobClient("resizedImage\_" + name).upload(new ByteArrayInputStream(output.toByteArray()), output.size());  context.getLogger().info("이미지가 input 컨테이너 Blob 저장소에 저장되었습니다. 파일명 :"+name);  context.getLogger().info("Original Image Size: " + originalWidth + "x" + originalHeight);  context.getLogger().info("Resized Image Size: " + resizedWidth + "x" + resizedHeight);    } catch (Exception e) {  // TODO: handle exception  context.getLogger().warning("Error processing image: " + e.getMessage());  e.printStackTrace(); // 예외 스택 트레이스 출력  }  } |

## BlobTrigger 이미지 리사이징 테스트

1. 기존에 Function이 동작 중이라면 Ctrl+c를 눌러 작업을 종료한 다음 mvn 명령어를 통해 build 후 실행시킵니다.

|  |
| --- |
| Build : mvn clean package |
| Function 실행 : mvn azure-functions:run |

1. Azure Portal에 접속하여 input컨테이너에 이미지를 업로드 후 리사이징 처리되는 로그를 확인합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Azure Portal에 접속하여 input컨테이너와 output컨테이너에 업로드 된 이미지 크기를 확인합니다. [완료]
   * input 컨테이너

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* + output 컨테이너

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

## HttpTrigger, BlobTrigger 배포

1. Ctrl + Shift + P를 눌러 팔레트를 열어 Azure Functions: Create Function App in Azure…를 선택합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 배포대상 구독을 선택합니다.

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Azure Portal에 배포될 Functions리소스명을 입력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 로컬 JDK버전에 맞는 버전을 선택합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Korea Central로 배포 리전을 선택합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 터미널 로그를 통해 배포 진행과정을 확인합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. WORKSPACE의 Deploy버튼을 클릭해서 “Deploy to Function App”을 클릭합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. Function 생성시 선택했던 구독을 다시 선택합니다.

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 생성한 Function 리소스를 선택합니다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Deploy를 클릭하여 기존 생성한 리소스에 덮어써서 배포를 진행합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. 배포 진행과정을 모니터링하면서 완료를 기다립니다.
   * 터미널 로그를 통해 build과정을 모니터링합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + 해당 구독에 생성된 리소스 중 Function App을 열어 배포여부를 관찰합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 배포가 진행되면서 Azure Portal에 새로 생성된 리소스 리스트를 확인합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 리소스 리스트 중 blob트리거를 배포하면서 생성된 스토리지 계정(holfunctionsjava\*\*\*\*\*)에 input컨테이너와 output컨테이너를 생성합니다.

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

1. 생성한 Functions리소스로 접근하여 생성된 트리거 리스트를 확인합니다.
   1. HOL-Functions-Java를 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 1. 생성한 트리거 리스트를 확인합니다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

## Azure Portal Blob트리거 테스트

1. 트리거 리스트 중 BlobTriggerJava를 클릭합니다.

텍스트, 소프트웨어, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 모니터를 클릭해 로그를 지속적으로 확인합니다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 다른 브라우저 창을 열어 스토리지 계정(holfunctionsjava107d0b)에서 input 컨테이너를 클릭합니다.

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

1. 업로드를 클릭해서 이미지를 업로드 테스트를 진행합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 파일 찾아보기를 클릭해서 테스트할 이미지를 선택 후 업로드를 클릭합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 이미지 업로드 완료 후 실행해 놓은 BlobTriggerJava 모니터를 통해 파일 리사이징 로그를 관찰합니다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

1. 스토리지 계정(holfunctionsjava107d0b)에서 output 컨테이너에 리사이징 된 이미지를 다운로드 받아 실제 이미지 사이즈에 변화가 있는지 확인합니다. [완료]

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

# Lab 4 – Python Functions 생성 및 실행

## MySQL Database 생성

1. Azure Portal에 접속 후 MySQL을 검색 후 “Azure Database for MySQL 유동 서버”를 선택합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. “+만들기”를 클릭해서 “유연한 서버”를 클릭하여 리소스 생성 작업을 진행합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. “기본” 탭에서 아래 내용을 참고하여 입력 후 나머지는 기본을 선택해서 작성합니다. (이미지 참고)
   * 1. 리소스 그룹 : 기존 스토리지 계정을 생성 시 추가된 리소스그룹을 선택합니다.
     2. 서버 이름 : 사용할 Database명을 입력합니다.
     3. 지역 : Korea Central을 선택합니다.
     4. 인증 방법 : MySQL 인증만을 선택합니다.
     5. 관리자 사용자 이름 : 사용할 DB계정을 입력합니다.
     6. 암호 : 사용할 DB 비밀번호를 입력합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

작성 후 “다음: 네트워킹” 버튼을 클릭합니다.

1. “네트워크” 탭에서 “Azure 내의 모든 Azure 서비스의 이 서버에 대한 퍼블릭 액세스 허용”에 체크를 한 후 “현재 클라이언트 IP 주소 추가”를 눌러 IP를 추가합니다.

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

“검토 + 만들기”를 클릭하여 다음으로 넘어갑니다.

1. “검토 + 만들기” 탭에서 “만들기”를 클릭해서 리소스 생성 작업을 진행합니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 배포가 완료되면 “리소스로 이동”을 클릭해서 개요로 이동합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. “서버 매개 변수”로 이동해서 “transport”를 검색한 다음 값을 “OFF”로 수정 후 “저장”을 클릭해서 수정된 매개 변수를 저장합니다. [완료]

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

## QueueTrigger 생성

1. 터미널에서 기존 경로와 다른 새로운 경로를 생성 후 아래 명령어를 통해 새로운 작업환경을 구성합니다.

|  |
| --- |
| cd : 새로운 경로로 이동  code . : 새 작업환경 열기 |

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Java에서 Function을 추가했던 것과 마찬가지로 아래 이미지에서 보이는 “Create Function” 버튼을 클릭합니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

1. 현재 디렉토리를 선택 후 엔터를 입력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Python을 선택 후 엔터를 입력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Skip virtual environment를 선택해서 넘어갑니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Azure Queue Storage Trigger를 선택합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 사용할 트리거의 대표 이름을 입력 후 엔터를 입력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Create new local app setting을 선택합니다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 사용할 구독을 선택합니다.
   * 단, 기존에 스토리지 계정이 배포된 구독을 선택 합니다.

스크린샷, 텍스트, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

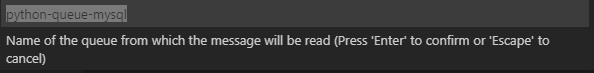
자동 생성된 설명

1. 로컬에서 Java Function작업시 사용했던 스토리지계정을 선택합니다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 스토리지 계정에 생성될 큐에 대한 명칭을 지어줍니다.
   * python-queue-mysql



1. Python코드로 이루어진 QueueTrigger생성이 완료되었습니다.[완료]

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

## QueueTrigger 로컬환경 실행

* Python 트리거 실행 명령어

|  |
| --- |
| func start |

1. 트리거를 실행 시 아래 이미지와 같은 에러가 발생하는 경우 [하단 이미지 참조]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + QueueTrigger 실행 전 host.json파일에서 이미지에 표시된 버전과 동일한 버전으로 수정합니다.
  + 기존 버전 > "[3.\*, 3.9.0)" 버전으로 수정

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* + 수정 후 재실행

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. function.json파일 수정
   * function.json파일에서 connection이 “functionsholst\_STORAGE”로 되어있으나 배포 시 AzureWebJobsStorage에서 스토리지 연결 문자열이 저장되기 때문에 connection을 AzureWebJobsStorage로 미리 변경합니다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. local.setting.json파일 수정
   * 로컬에서 테스트 진행 시 local.setting.json에서 환경변수를 사용하기 때문에 비어있는 AzureWebJobsStorage변수에 functionsholst\_STORAGE에 저장된 스토리지 연결 문자열을 똑같이 복사해서 넣어줍니다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 위 세개의 파일을 수정 후 터미널에서 다시 트리거를 실행합니다.
   * Python 트리거 실행 명령어

|  |
| --- |
| func start |

1. 기존 스토리지 계정에서 “큐”를 클릭한 후 “+큐”를 눌러 QueueTrigger 생성 시 지어준 큐 이름(python-queue-mysql)으로 큐를 생성합니다.

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

1. 해당 큐를 클릭해서 “+메시지” 버튼을 누르고 메시지를 추가합니다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

1. 메시지 추가 시 아래 이미지와 같은 python코드에 의해 생성된 로그를 확인할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

## 데이터 인입 확인을 위한 http트리거 추가

1. “Create Function” 버튼을 클릭합니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

1. 현재 디렉토리를 선택 후 엔터를 입력합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. http트리거에 사용할 이름을 작성합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Anonymous를 선택해서 http트리거 생성을 완료합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. http트리거 생성이 완료되었습니다.

텍스트, 소프트웨어, 폰트, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. http트리거 생성 후 \_\_init\_\_.py파일의 코드를 다음과 같이 수정합니다.
   * + 데이터베이스 생성 시 사용되는 정보들을 아래 코드의 주석(# 데이터베이스 정보)에 있는 값에 동일하게 입력합니다.

|  |
| --- |
| import logging  import mysql.connector  import azure.functions as func  def main(req: func.HttpRequest) -> func.HttpResponse:      logging.info('Python HTTP trigger function processed a request.')      # 데이터베이스 정보      host = "<<DB HOST NAME>>"      port = 3306      user = "<<DB USER NAME>>"      password = "<<DB PASSWORD>>"      database = "<<DB NAME>>"      table = "<<DB TABLE NAME>>"      # Data SELECT      rows = select\_table(host, port, user, password, database, table)      return func.HttpResponse("\n".join([str(item[0])+", "+str(item[2])+"\n" for item in rows]),status\_code=200)  def select\_table(host, port, user, password, database, table):      logging.info("SELECT DATA")      # MySQL 데이터베이스에 연결      conn = mysql.connector.connect(          host=host,          port=port,          user=user,          password=password,          database=database,      )      cursor = conn.cursor()      # 메시지 데이터를 MySQL에 삽입      select\_query = f"""SELECT \* FROM {table}"""      cursor.execute(select\_query)        # SELECT 쿼리의 결과를 반환      rows = cursor.fetchall()        # 연결 종료      cursor.close()      conn.close()      logging.info(rows)      return rows |

## 코드 수정 및 테스트

1. 라이브러리 설치를 위해 requirements.txt파일을 다음과 같이 수정합니다.

|  |
| --- |
| azure-functions  azure-storage-queue  mysql-connector-python |

1. Requirements.txt파일에 추가된 라이브러리를 설치하기 위해 터미널에서 아래에 있는 명령어를 실행합니다.

|  |
| --- |
| pip install -r requirements.txt |

1. 다음 이미지와 같이 QueueTrigger폴더를 열어 \_\_init\_\_.py에 작성된 파이썬 코드를 다음과 같이 수정합니다.

|  |
| --- |
| import logging  import mysql.connector  from azure.functions import QueueMessage  def main(msg: QueueMessage) -> None:      logging.info('Python queue trigger function processed a queue item: %s', msg.get\_body().decode('utf-8'))        # message 정보 읽어오기      message\_body = msg.get\_body().decode('utf-8')      message\_id = msg.id      message\_receipt = msg.pop\_receipt      logging.info("message context : %s", message\_body)      logging.info("message\_id : %s", message\_id)      logging.info("message\_receipt : %s", message\_receipt)      # 데이터베이스 정보      host = "<<DB HOST NAME>>"      port = 3306      user = "<<DB USER NAME>>"      password = "<<DB PASSWORD>>"      database = "<<DB NAME>>"      table = "<<DB TABLE NAME>>"      # 데이터베이스 생성      create\_database(host, port, user, password, database)      # 테이블 생성      create\_table(host, port, user, password, database, table)      # Data INSERT      insert\_table(host, port, user, password, database, table, message\_id, message\_body)      # Data SELECT      select\_table(host, port, user, password, database, table)  def create\_database(host, port, user, password, database):      # MySQL 데이터베이스에 연결      logging.info("CREATE DATABASE")      conn = mysql.connector.connect(          host=host,          port=port,          user=user,          password=password,      )      cursor = conn.cursor()      # 데이터베이스 생성 쿼리 실행      create\_database\_query = f"CREATE DATABASE IF NOT EXISTS {database}"      cursor.execute(create\_database\_query)      # 연결 종료      cursor.close()      conn.close()  def create\_table(host, port, user, password, database, table\_name):      logging.info("CREATE TABLE")      # MySQL 데이터베이스에 연결      conn = mysql.connector.connect(          host=host,          port=port,          user=user,          password=password,          database=database,      )      cursor = conn.cursor()      # 테이블이 존재하는지 확인하고 없으면 생성      cursor.execute(f"SHOW TABLES LIKE '{table\_name}'")      result = cursor.fetchone()      if not result:          # 테이블 생성 쿼리 실행          create\_table\_query = f"""          CREATE TABLE IF NOT EXISTS {table\_name} (              id INT AUTO\_INCREMENT PRIMARY KEY,              message\_id VARCHAR(255),              message\_body TEXT          )          """          cursor.execute(create\_table\_query)          logging.info("테이블 생성이 완료되었습니다.")      # 연결 종료      cursor.close()      conn.close()  def insert\_table(host, port, user, password, database, table, message\_id, message\_body):      logging.info("INSERT DATA")      # MySQL 데이터베이스에 연결      conn = mysql.connector.connect(          host=host,          port=port,          user=user,          password=password,          database=database,      )      cursor = conn.cursor()      # 메시지 데이터를 MySQL에 삽입      insert\_query = f"""INSERT INTO {table} (message\_id, message\_body) VALUES (%s, %s)"""      insert\_values = (message\_id, message\_body)      cursor.execute(insert\_query, insert\_values)      conn.commit()      # 연결 종료      cursor.close()      conn.close()  def select\_table(host, port, user, password, database, table):      logging.info("SELECT DATA")      # MySQL 데이터베이스에 연결      conn = mysql.connector.connect(          host=host,          port=port,          user=user,          password=password,          database=database,      )      cursor = conn.cursor()      # 메시지 데이터를 MySQL에 삽입      select\_query = f"""SELECT \* FROM {table}"""      cursor.execute(select\_query)        # SELECT 쿼리의 결과를 반환      rows = cursor.fetchall()        # 연결 종료      cursor.close()      conn.close()      logging.info(rows) |

1. 터미널에서 다시 트리거를 실행합니다.
   * Python 트리거 실행 명령어

|  |
| --- |
| func start |

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

## QueueTrigger 테스트

1. 개요에 있는 “서버 이름”의 URL을 복사하여 수정된 파이썬 코드에 있는 DB정보의 host에 붙여 넣고 다른 정보도 함께 입력합니다.
   1. 서버 이름 복사

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

* 1. http트리거와 QueueTrigger에 있는 \_\_init\_\_.py의 데이터베이스 정보 host변수에 붙여 넣고 나머지도 MySQL 생성 시 사용했던 정보들로 입력 후 function을 로컬에서 실행합니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

1. 큐(python-queue-mysql)로 돌아가 다시 한번 “+메시지 추가”를 클릭하여 메시지를 추가 후 터미널에서 해당 로그를 확인합니다.
   1. 큐에 메시지를 추가합니다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

* 1. 터미널 로그를 확인. [완료]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

큐에 메시지를 추가하면서 해당 데이터베이스에 정상적으로 데이터 Insert가 확인됩니다.

# Lab 5 – Python Functions 배포

## Azure Portal Function생성

1. Ctrl+Shift+P을 열어 팔레트를 켠 후 팔레트에서 아래 이미지에 표시된 부분을 선택합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 기존과 같은 구독을 선택합니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 배포할 리소스 이름을 작성합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 로컬 환경과 비슷한 버전을 선택합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 해당 리전을 Korea Central로 선택합니다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 해당 구독 하위 디렉토리에서 생성한 Function이 조회되는지 확인합니다. [완료]

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

## Python QueueTrigger 배포 및 데이터 확인

1. 다음 화면에서 Deploy버튼을 눌러 이미지에 표시된 부분을 클릭합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. 기존과 동일한 구독을 선택합니다.

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. Task1에서 생성한 리소스를 선택합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Deploy버튼을 눌러 배포를 진행합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. 새 브라우저를 열어 기존에 작업한 Function을 생성하면서 자동으로 생성된 스토리지에 같은 이름의 큐를 생성합니다.

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

1. 해당 큐로 진입해 메시지를 추가합니다.

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Function화면으로 전환해서 생성한 Http트리거를 클릭합니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. “함수URL가져오기”를 클릭 후 “클립보드 복사”버튼을 눌러 URL을 복사한 다음 테스트할 새 브라우저에 URL을 붙여넣어 큐에 추가한 메시지 리스트를 확인합니다.

텍스트, 소프트웨어, 폰트, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 완료

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명