**Java unzip azure function**

**Introduction**

This is an example document to show how to upload a zip file to azure storage blob and unzip using java function on message received from message queue (Queue Trigger).

Azure function can allow us to write function with Node.js, C#, F#, Python, PHP, and even Java.

Charges are applicable only when functions are used or triggered.

Azure queue re- tries for 5 times in case of message delivery failed. At 6th time it will be added to poison queue. We can manipulate it based on our requirement.

**Prerequisites**

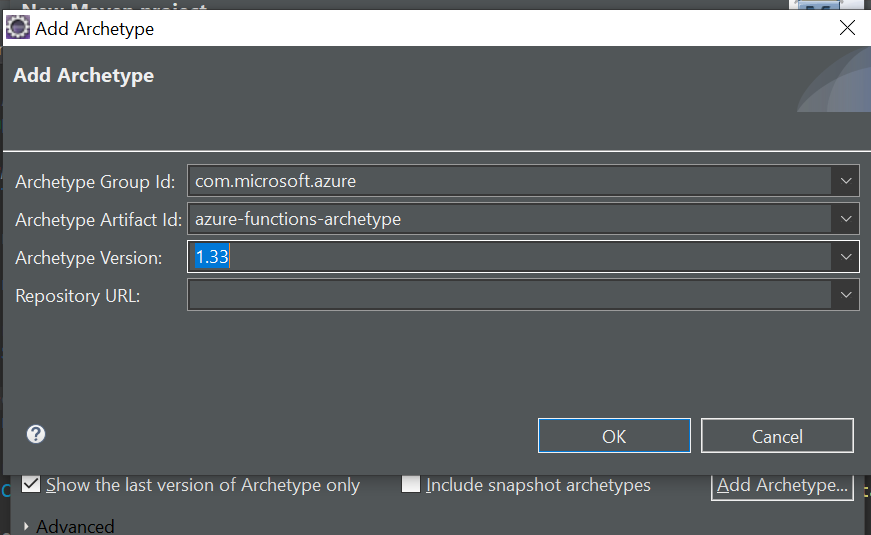
1. Install [Azure functions core tool](https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local#v2).
2. Install [Azure CLI](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
3. Java developer kit version 8 or 11.
4. Apache Maven version 3.0 or above.

**Prerequisite check**

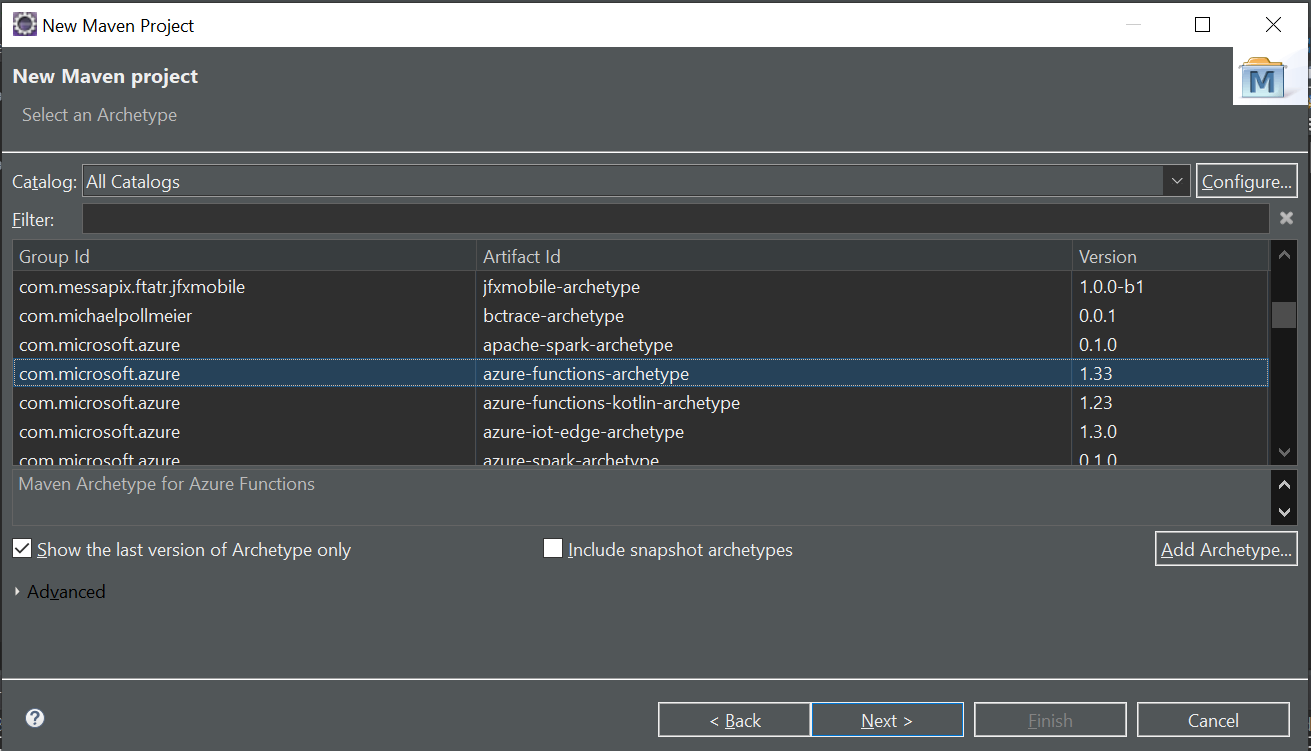
1. In a terminal or command window, run func --version to check that the Azure Functions Core Tools are version 3.x.
2. Run az --version to check that the Azure CLI version is 2.4 or later.
3. Run az login to sign in to Azure and verify an active subscription.

**Steps to create azure function with Java**

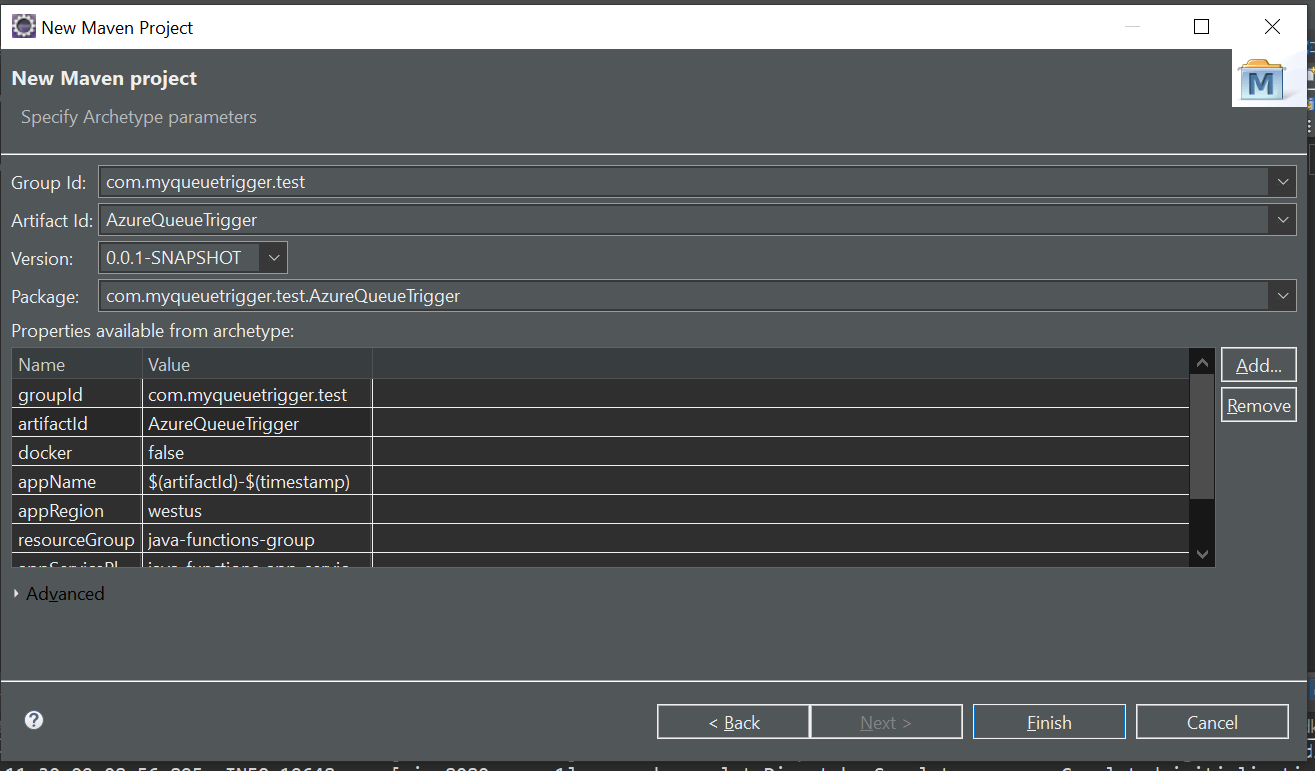
1. Create a new Maven project and add archetype



1. Double click on Azure functions archetype



1. Enter following details and finish



1. In POM.xml file update the function name to “AzureQueueTriggerTest” and add following dependency

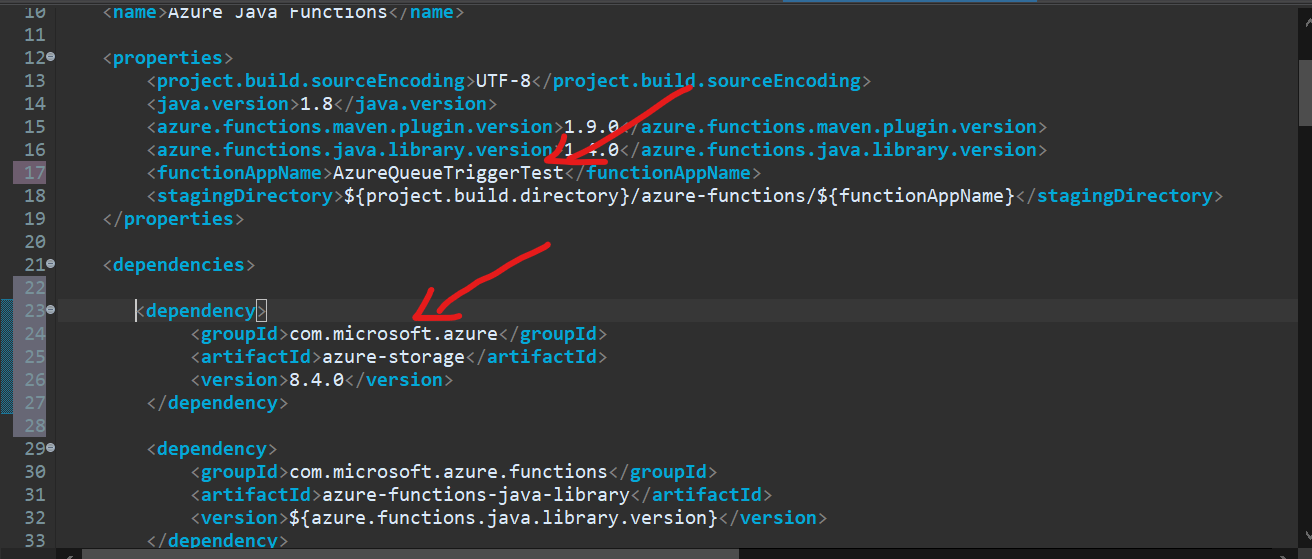
<**dependency**>

<**groupId**>com.microsoft.azure</**groupId**>

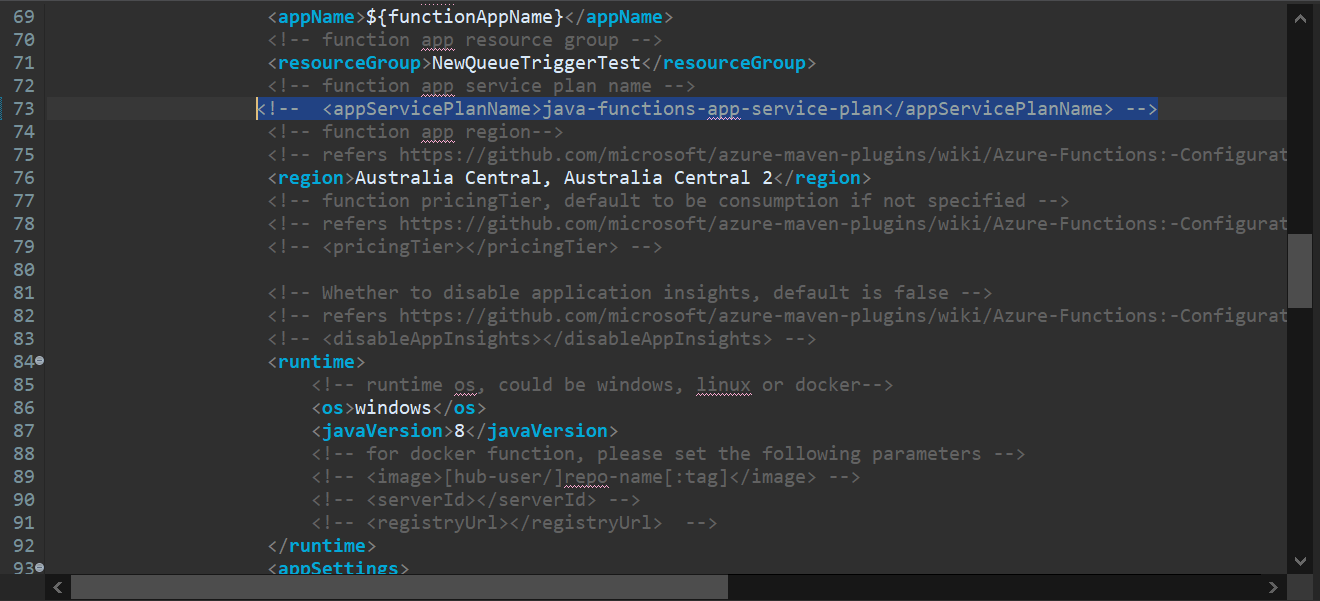
<**artifactId**>azure-storage</**artifactId**>

<**version**>8.4.0</**version**>

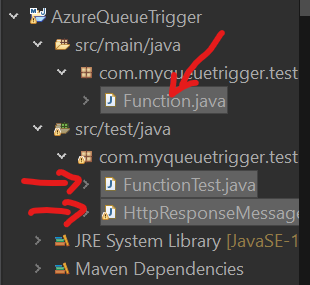
</**dependency**>



1. Add Resource group, Region, OS and comment app service plan



1. Delete the Function.java class and also delete content of test package



1. Create a java class with name “QueueTriggerTest.java” and paste following code. @QueueTrigger annotation which acts as listener message queue and gets triggered when message is entered to queue, we need to pass queueName, connection which is defined in lock.settings.json file dataType should be empty as we are using java Bean(Details.java).

@BlobInput annotation represents input to the method as we are providing zip file as input. We need to pass dataType as binary as we are using zip file, connection which is defined in lock.settings.json, path to the zip file on azure blob storage as I have file path in java Bean (Details.java) just need to pass variable name like “{Path}”

Zip file will be unzipped and uploaded to defined container on azure blob storage (Create container on azure pass the same name to containerName variable)

@FunctionName("queueprocessor")

public void run(

@QueueTrigger(name = "msg",

queueName = "queuetest",

dataType = "",

connection = "AzureWebJobsStorage") Details message,

final ExecutionContext executionContext,

@BlobInput(

name = "file",

dataType = "binary", connection = "AzureWebJobsStorage",

path = "{Path}") byte[] content

) {

executionContext.getLogger().info(message.getPath() + " :: " + content);

CloudStorageAccount storageAccount = null;

CloudBlobClient blobClient = null;

CloudBlobContainer container=null;

try {

String connectStr = "DefaultEndpointsProtocol=https;AccountName=queuelistener573996;AccountKey=4hGXZopXoN+8n3z42w+IPC2GDUAjRblMFfbETrJhIuG0I+KrrqJFYghad1BjtjGJxOYODzMBSkpXWoRUmjkZSg==;EndpointSuffix=core.windows.net"; // Use your account connection string

//unique name of the container

String containerName = "output";

// Parse the connection string and create a blob client to interact with Blob storage

storageAccount = CloudStorageAccount.parse(connectStr);

blobClient = storageAccount.createCloudBlobClient();

container = blobClient.getContainerReference(containerName);

container.createIfNotExists(BlobContainerPublicAccessType.CONTAINER, new BlobRequestOptions(), new OperationContext());

InputStream targetStream = new ByteArrayInputStream(content);

executionContext.getLogger().info("QueueTriger Input Stream : " + targetStream);

ZipInputStream zipIn = new ZipInputStream(targetStream);

ZipEntry zipEntry = zipIn.getNextEntry();

while(zipEntry != null) {

executionContext.getLogger().info("ZipEntry Name: " + zipEntry.getName());

//Getting a blob reference

CloudBlockBlob blob = container.getBlockBlobReference(zipEntry.getName());

ByteArrayOutputStream outputB = new ByteArrayOutputStream();

byte[] buf = new byte[1024];

int n;

while ((n = zipIn.read(buf, 0, 1024)) != -1) {

outputB.write(buf, 0, n);

}

// Upload to container

ByteArrayInputStream inputS = new ByteArrayInputStream(outputB.toByteArray());

blob.upload(inputS, inputS.available());

executionContext.getLogger().info("QueueTriger DONE: ");

zipEntry = zipIn.getNextEntry();

}

zipIn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

1. Create a java class with name “Details.java” add following code

private String id;

private String path;

public Details() {

super();

}

public Details(String id, String path) {

super();

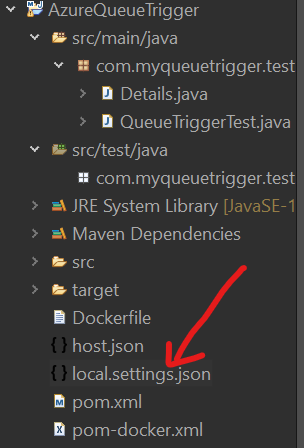
this.id = id;

this.path = path;

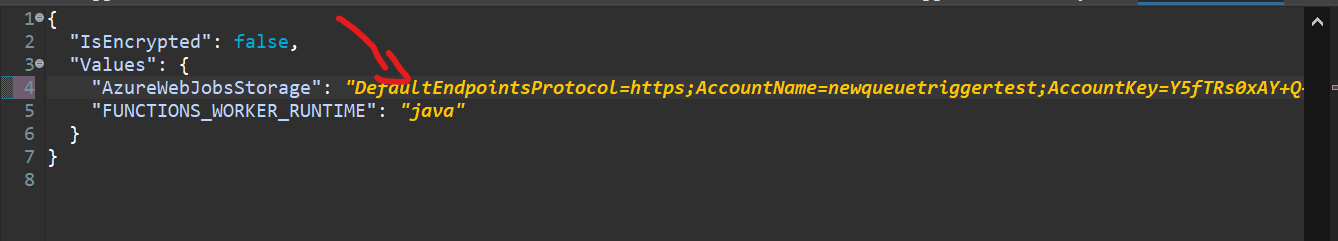
}

// Setters and getters

1. Go to lock.settings.json

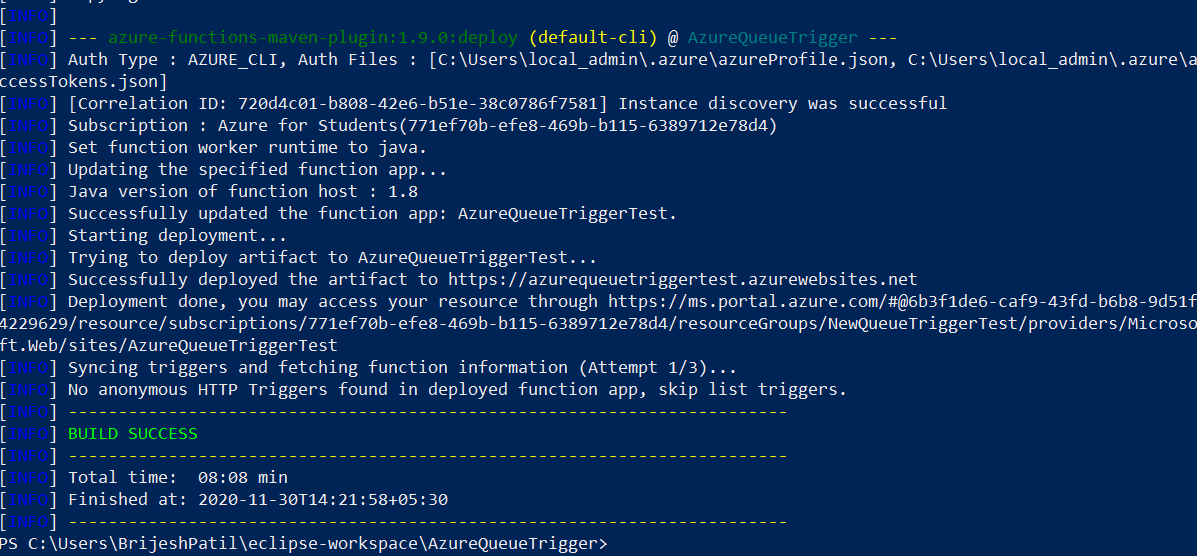


1. Add connection string from your azure blob storage account as shown below

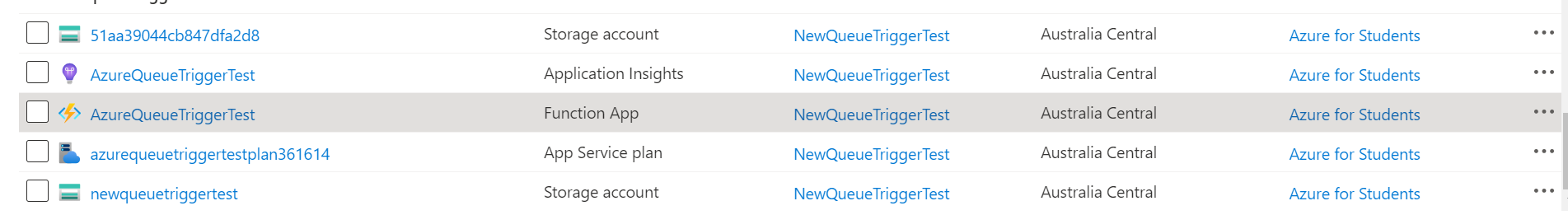


1. Open command prompt from project root location and run following command to deploy to azure

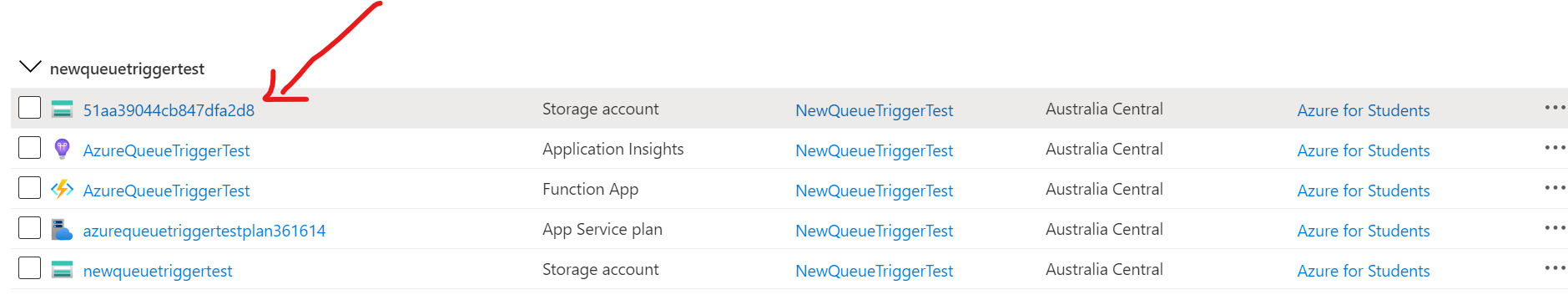
mvn install package azure-functions:deploy



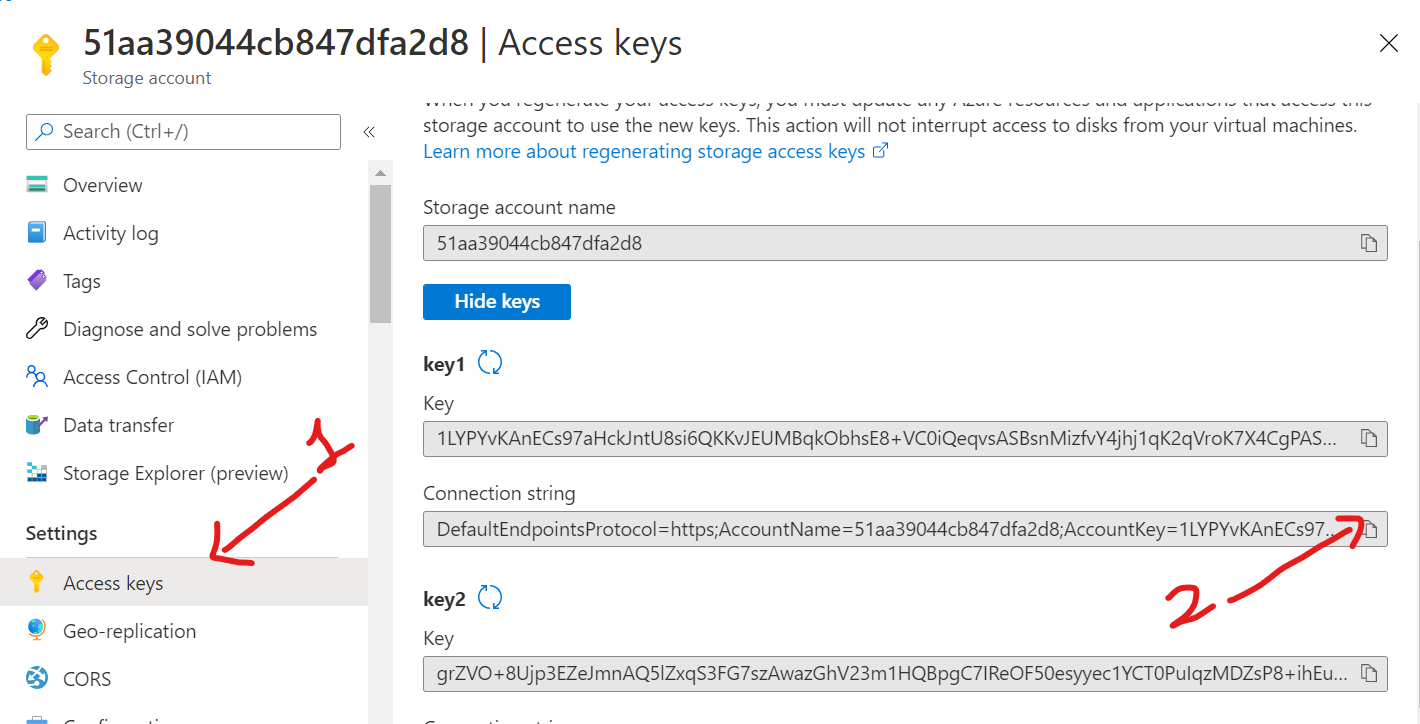
1. After deploying you can see function app on azure



1. Replace the connectStr variable with newly created Storage account connection string. Select below shown new storage account



1. Copy connection string



1. Replace AzureWebJobsStorage in lock.settings.json file with new connection string
2. Re-deploy

mvn install package azure-functions:deploy

**Create a spring boot application**

1. Create a simple spring boot application which will upload zip file to azure storage blob.
2. Add below dependencies

<dependency>

<groupId>com.azure</groupId>

<artifactId>azure-storage-blob</artifactId>

<version>12.6.0</version>

</dependency>

<dependency>

<groupId>com.azure</groupId>

<artifactId>azure-storage-queue</artifactId>

<version>12.6.0</version>

</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>${gson.version}</version>

</dependency>

1. Use below code to upload a multipart file or zip file and publish message on azure queue.

@PostMapping("/upload")

public String uploadFile(@RequestParam("file") MultipartFile file) throws IOException {

try {

String connectStr = "DefaultEndpointsProtocol=https;AccountName=51aa39044cb847dfa2d8;AccountKey=1LYPYvKAnECs97aHckJntU8si6QKKvJEUMBqkObhsE8+VC0iQeqvsASBsnMizfvY4jhj1qK2qVroK7X4CgPASQ==;EndpointSuffix=core.windows.net"; // Use your account connection string

// Create a BlobServiceClient object which will be used to create a container client

BlobServiceClient blobServiceClient = new BlobServiceClientBuilder().connectionString(connectStr).buildClient();

//Create a container on azure to which files will be uploaded

String containerName = "zipfiles";

// Create the container and return a container client object

BlobContainerClient containerClient = blobServiceClient.getBlobContainerClient(containerName);

// Get a reference to a blob

BlobClient blobClient = containerClient.getBlobClient(file.getOriginalFilename());

// Upload to container

blobClient.upload(file.getInputStream(), file.getSize(), true);

Details d = new Details("123", blobClient.getContainerName()+"/"+blobClient.getBlobName());

//Publish message on queue

addQueueMessage(connectStr, "queuetest", d);

return "Done";

} catch (Exception e) {

return e.getMessage();

}

}

public static void addQueueMessage(String connectStr, String queueName, Details message) throws QueueStorageException {

// Instantiate a QueueClient which will be

// used to create and manipulate the queue

QueueClient queueClient = new QueueClientBuilder()

.connectionString(connectStr)

.queueName(queueName)

.buildClient();

System.out.println("Adding message to the queue: " + message);

// Add a message to the queue

Gson gSon = new Gson();

String messageToSend = gSon.toJson(message);

System.out.println(messageToSend);

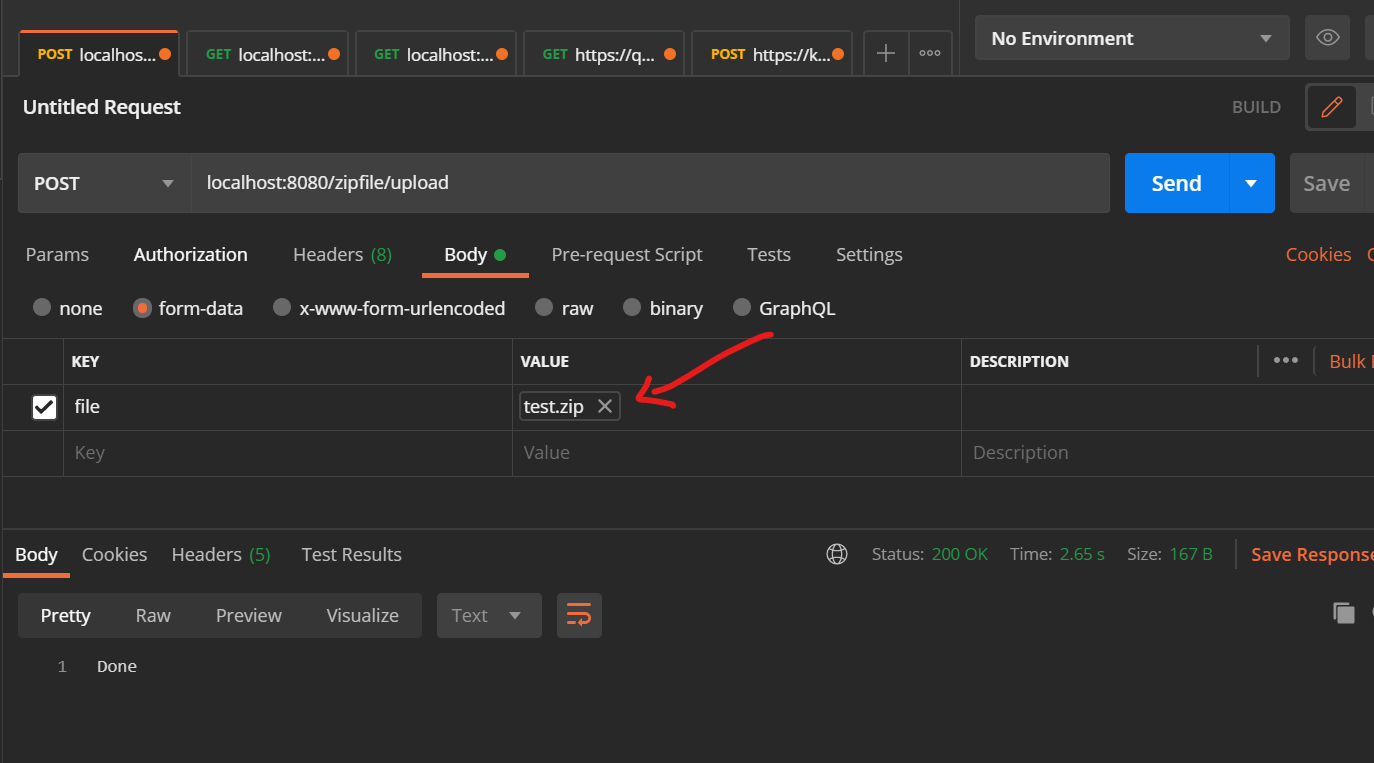
String encodedMsg = Base64.getEncoder().encodeToString(messageToSend.getBytes());

queueClient.sendMessage(encodedMsg);

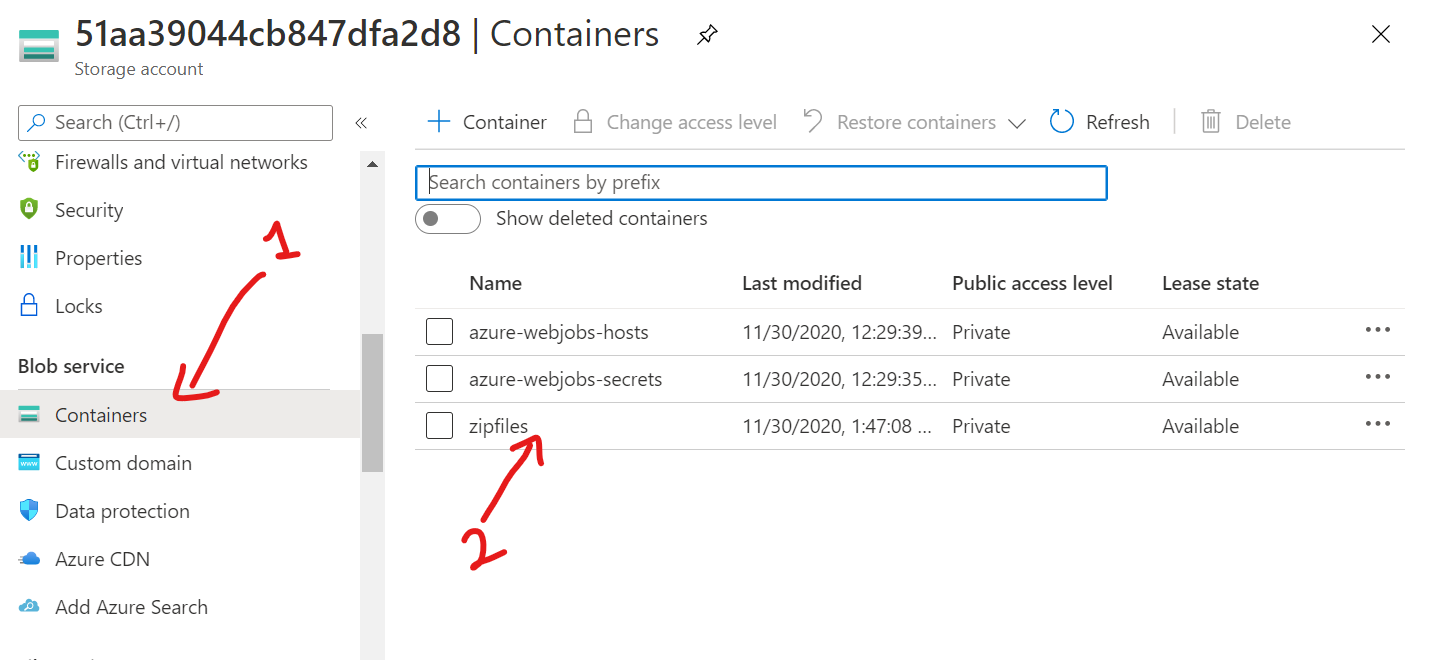
}

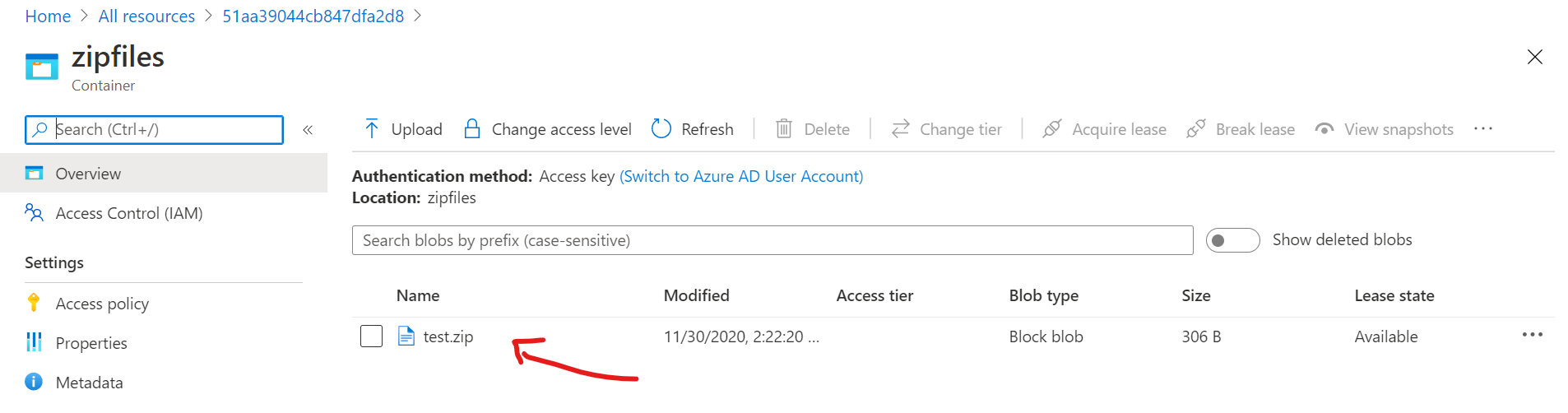
Now upload a file from spring boot application which will be extracted to specified output container

1. Upload zip file

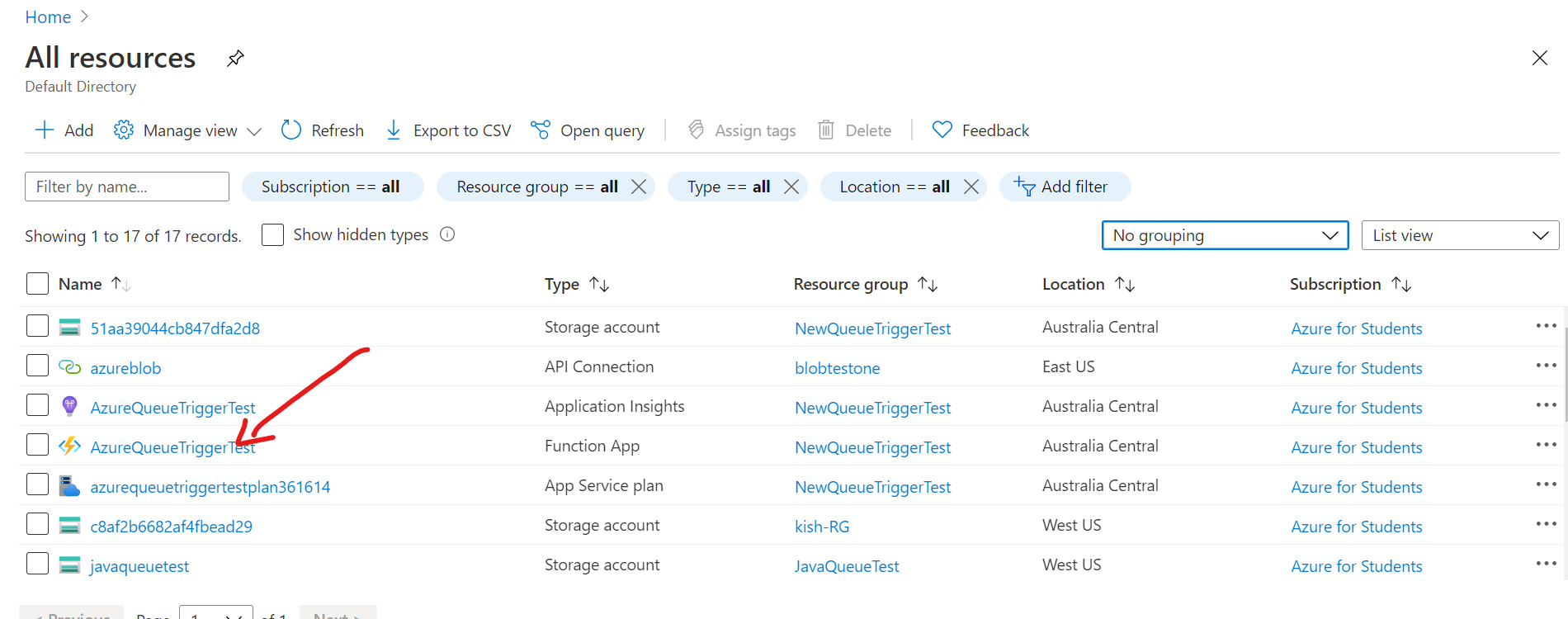


1. Go to azure storage container where you can see uploaded zip file

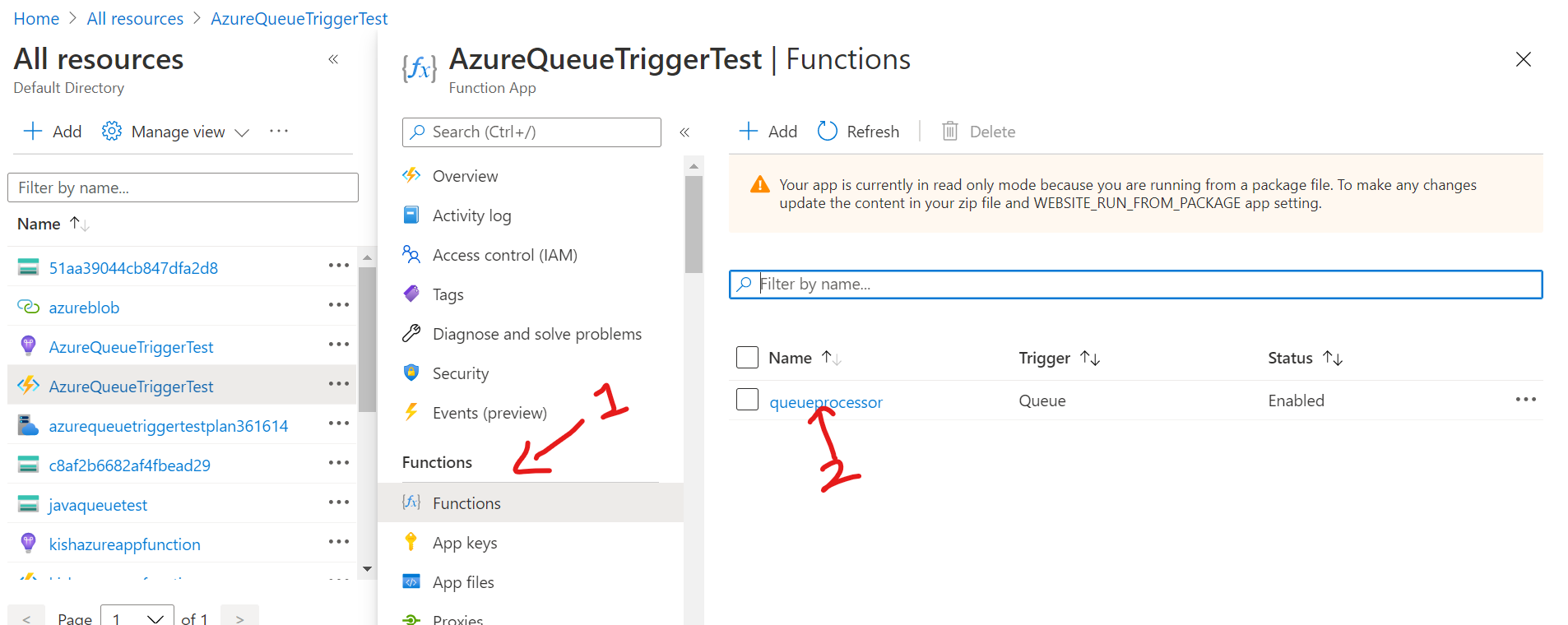




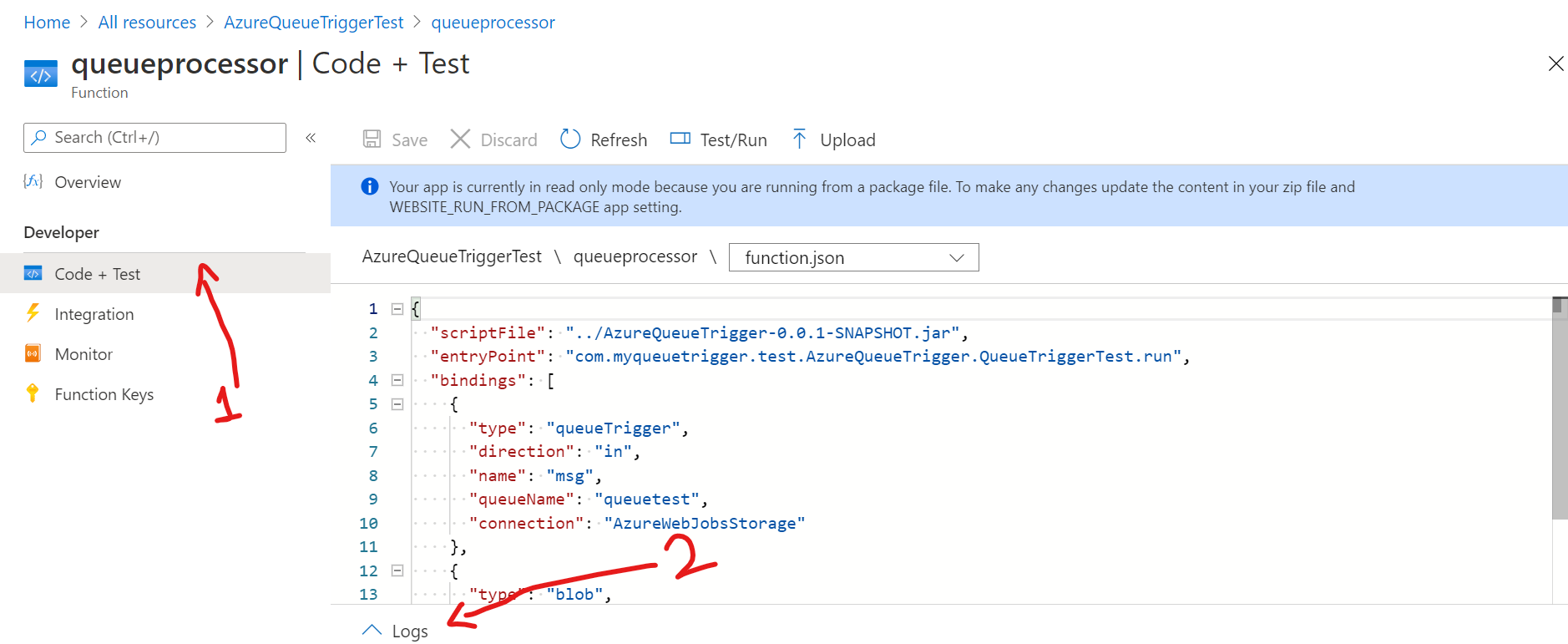
1. To see the logs on azure go to all resources look out for your function app as shown below



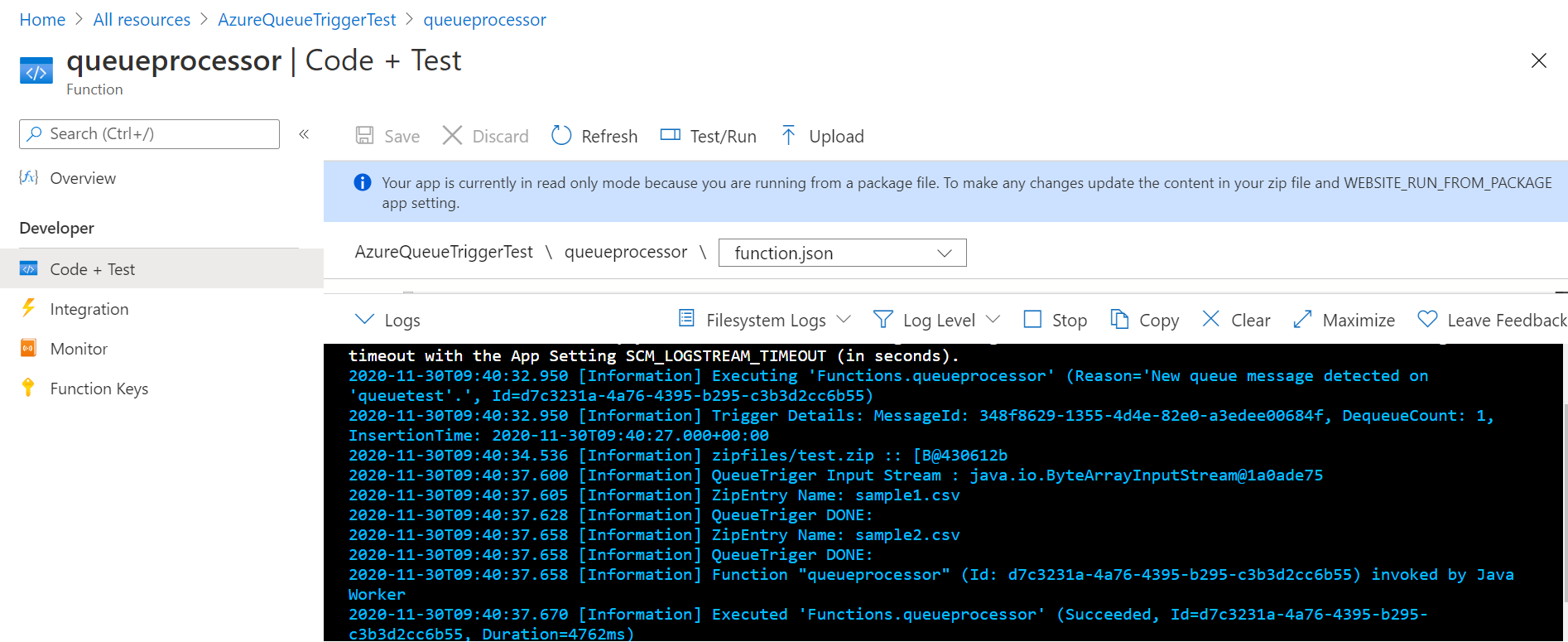
1. Click on Functions and select trigger function shown



1. Click on Code + Test then Logs



1. Now upload another file to see the logs as shown in below image



1. Go to output container where extracted files are stored

