Below is a practical example of an **Azure WebJob** written in Python that performs the following tasks:

1. Polls Azure Blob Storage for new files.
2. Processes each file (e.g., reads its contents or performs a transformation).
3. Sends the processed results to a webhook.

### **Prerequisites**

1. **Install dependencies**:

bash

Copy code

pip install azure-storage-blob requests

1. **Create an Azure Blob Storage account** and container.

### **WebJob Code**

Save the code as webjob.py.

python

Copy code

import os

import time

from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

import requests

# Configuration

AZURE\_STORAGE\_CONNECTION\_STRING = os.getenv("AZURE\_STORAGE\_CONNECTION\_STRING", "<your\_connection\_string>")

CONTAINER\_NAME = os.getenv("CONTAINER\_NAME", "uploads")

WEBHOOK\_URL = os.getenv("WEBHOOK\_URL", "http://your-backend.com/webhook")

PROCESSING\_INTERVAL = int(os.getenv("PROCESSING\_INTERVAL", 30)) # Seconds

# Initialize Blob Service Client

blob\_service\_client = BlobServiceClient.from\_connection\_string(AZURE\_STORAGE\_CONNECTION\_STRING)

container\_client = blob\_service\_client.get\_container\_client(CONTAINER\_NAME)

def process\_blob(blob\_name, blob\_data):

"""

Simulate processing the blob. You can replace this with your actual processing logic.

"""

print(f"Processing blob: {blob\_name}")

# Example: Convert the blob data to uppercase (as a placeholder for actual logic)

processed\_data = blob\_data.decode('utf-8').upper()

return processed\_data

def main():

print("Starting Azure WebJob...")

while True:

try:

print("Checking for new blobs...")

blobs = container\_client.list\_blobs()

for blob in blobs:

blob\_name = blob.name

print(f"Found blob: {blob\_name}")

# Download the blob

blob\_client = container\_client.get\_blob\_client(blob\_name)

downloaded\_blob = blob\_client.download\_blob()

blob\_data = downloaded\_blob.readall()

# Process the blob

processed\_data = process\_blob(blob\_name, blob\_data)

# Trigger the webhook

webhook\_payload = {

"filename": blob\_name,

"processed\_data": processed\_data,

"status": "success",

}

response = requests.post(WEBHOOK\_URL, json=webhook\_payload)

print(f"Webhook response: {response.status\_code}, {response.text}")

# Optionally delete the blob after processing

print(f"Deleting blob: {blob\_name}")

container\_client.delete\_blob(blob\_name)

except Exception as e:

print(f"Error: {e}")

# Wait before polling again

print(f"Waiting for {PROCESSING\_INTERVAL} seconds...")

time.sleep(PROCESSING\_INTERVAL)

if \_\_name\_\_ == "\_\_main\_\_":

main()

### **Steps to Deploy and Run the WebJob**

#### 1. **Set Up Azure WebJobs**

* Create an **Azure Web App** in the Azure portal.
* Navigate to the **WebJobs** section under the Web App's settings.
* Add a new WebJob:
  + **Name**: FileProcessor
  + **File**: Upload a .zip file containing the webjob.py script and a requirements.txt file.
  + **Run Mode**: Continuous.

#### 2. **Environment Variables**

Set the following environment variables in the Azure Web App's **Configuration** settings:

* AZURE\_STORAGE\_CONNECTION\_STRING: Your Azure Blob Storage connection string.
* CONTAINER\_NAME: The name of the blob container (e.g., uploads).
* WEBHOOK\_URL: The URL of your backend webhook endpoint.
* PROCESSING\_INTERVAL: The polling interval (default: 30 seconds).

#### 3. **Example: Webhook Endpoint**

Ensure the backend webhook endpoint (e.g., http://your-backend.com/webhook) is ready to receive POST requests. Example code in Quart:

python

Copy code

@app.route("/webhook", methods=["POST"])

async def webhook():

data = await request.json

print(f"Webhook triggered with data: {data}")

return jsonify({"message": "Webhook received", "status": "ok"}), 200

### **How It Works**

1. The WebJob continuously polls the Azure Blob Storage container for new files.
2. For each file:
   * Downloads the file.
   * Processes the file content (e.g., converts it to uppercase).
   * Sends the processed data to the backend using a webhook.
3. Deletes the file from the blob container after successful processing.
4. Waits for the defined interval before polling again.

This approach is efficient and well-suited for asynchronous file processing workflows. You can adapt the process\_blob function to implement your specific processing logic.

------------------------------------------------------------------------------

**Replace local folder processing with azure blob:** (**refer file\_data for processing files passed from frontend**):

from quart import Quart, jsonify

from azure.storage.blob import BlobServiceClient

import os

app = Quart(\_\_name\_\_)

# Set up Azure Blob Storage

CONNECTION\_STRING = os.getenv("AZURE\_STORAGE\_CONNECTION\_STRING")

CONTAINER\_NAME = os.getenv("AZURE\_CONTAINER\_NAME")

blob\_service\_client = BlobServiceClient.from\_connection\_string(CONNECTION\_STRING)

container\_client = blob\_service\_client.get\_container\_client(CONTAINER\_NAME)

@app.route('/process-files')

async def process\_files():

try:

blobs = container\_client.list\_blobs()

file\_data = []

for blob in blobs:

# Download blob content

blob\_client = container\_client.get\_blob\_client(blob)

stream = blob\_client.download\_blob()

content = stream.readall()

# Process the content as needed

# For example, adding content metadata to the response

file\_data.append({

"name": blob.name,

"size": blob.size,

"content": content.decode('utf-8')[:100], # Preview first 100 chars

})

return jsonify({"files": file\_data})

except Exception as e:

return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

**Upload files to blob and message queue creation:**

from quart import Quart, request, jsonify

from azure.storage.blob import BlobServiceClient

from azure.storage.queue import QueueServiceClient

import json

app = Quart(\_\_name\_\_)

# Azure Blob and Queue Configuration

BLOB\_CONNECTION\_STRING = "<your\_blob\_storage\_connection\_string>"

QUEUE\_CONNECTION\_STRING = https://sapocopenai.queue.core.windows.net/file-processing-queue

BLOB\_CONTAINER\_NAME = "uploads" # Change to your Blob container name

# Function to upload files to Blob Storage

def upload\_file\_to\_blob(file, blob\_name):

blob\_service\_client = BlobServiceClient.from\_connection\_string(BLOB\_CONNECTION\_STRING)

blob\_client = blob\_service\_client.get\_blob\_client(container=BLOB\_CONTAINER\_NAME, blob=blob\_name)

blob\_client.upload\_blob(file, overwrite=True)

return f"https://{blob\_service\_client.account\_name}.blob.core.windows.net/{BLOB\_CONTAINER\_NAME}/{blob\_name}"

# Function to enqueue a message to Azure Queue

def enqueue\_message\_to\_queue(file\_url, user\_id, session\_id):

queue\_service = QueueServiceClient.from\_connection\_string(QUEUE\_CONNECTION\_STRING)

queue\_client = queue\_service.get\_queue\_client(queue="file-processing-queue")

message = {

"file\_url": file\_url,

"user\_id": user\_id,

"session\_id": session\_id

}

queue\_client.send\_message(json.dumps(message))

print(f"Message enqueued for user: {user\_id}, session: {session\_id}, file\_url: {file\_url}")

# API Endpoint for File Upload

@app.route('/upload', methods=['POST'])

async def upload():

data = await request.form

file = await (await request.files).get('file') # File upload

user\_id = data.get("user\_id")

session\_id = data.get("session\_id")

# Upload the file to Azure Blob Storage

blob\_name = f"{session\_id}/{file.filename}" # Organize by session

file\_url = upload\_file\_to\_blob(file, blob\_name)

# Enqueue the file URL and metadata to Azure Queue Storage

enqueue\_message\_to\_queue(file\_url, user\_id, session\_id)

return jsonify({"status": "Enqueued for processing", "file\_url": file\_url})

if \_\_name\_\_ == '\_\_main\_\_':

app.run()