CDW Documentation

Deploy Face Blur Script to Azure Function App

Deploy Face Blur Script to Azure Function App

Prerequisites

Install Azure CLI # Windows curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

macOS brew install azure-cli

Or download from: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli

Install Azure Functions Core Tools

```
# Windows (via npm)
npm install -g azure-functions-core-tools@4 --unsafe-perm true
```

```
# macOS
brew tap azure/functions
brew install azure-functions-core-tools@4
```

```
# Linux
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor >
microsoft.gpg
sudo mv microsoft.gpg /etc/apt/trusted.gpg.d/microsoft.gpg
sudo sh -c 'echo "deb [arch=amd64]
https://packages.microsoft.com/repos/microsoft-ubuntu-$(lsb_release -cs)-
prod $(lsb_release -cs) main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-get update
sudo apt-get install azure-functions-core-tools-4
```

Step 1: Login to Azure

```
az login


Step 2: Set Your Subscription (if you have multiple)
bash# List subscriptions
az account list --output table

# Set active subscription

az account set --subscription "Your-Subscription-Name-Or-ID"
```

Step 3: Create Resource Group (if needed) bash

```
az group create --name myResourceGroup --location eastus
```

Step 4: Create Storage Account (for Function App) bash

```
az storage account create \
  --name myfunctionstorageacct \
  --location eastus \
  --resource-group myResourceGroup \
  --sku Standard_LRS
```

Step 5: Create Function App bash

```
az functionapp create \
  --resource-group myResourceGroup \
  --consumption-plan-location eastus \
  --runtime python \
  --runtime-version 3.9 \
  --functions-version 4 \
  --name myFaceBlurFunctionApp \
  --storage-account myfunctionstorageacct \
  --os-type linux
```

Step 6: Prepare Your Project Structure Create the following folder structure: face-blur-function/ ├── requirements.txt ├── host.json ├── local.settings.json ├── FaceBlurTimer/ │ ├── init.py │ └── function.json └── shared/

```
  └── face_blur_processor.py
```

Step 7: Create Project Files requirements.txt

```
txtazure-functions
azure-storage-blob
requests
python-dotenv
Pillow
schedule
```

host.json

```
{
  "version": "2.0",
  "logging": {
    "applicationInsights": {
      "samplingSettings": {
        "isEnabled": true,
        "excludedTypes": "Request"
      }
    }
  },
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle",
    "version": "[2.*, 3.0.0)"
  },
  "functionTimeout": "00:10:00"
```

```
}
```

local.settings.json

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "your_function_storage_connection_string",
    "FUNCTIONS_WORKER_RUNTIME": "python",
    "AZURE_STORAGE_CONNECTION_STRING":
"your_blob_storage_connection_string",
    "AZURE_CONTAINER_NAME": "your_source_container_name",
    "WEBAPP_URL": "https://your-webapp.com/blur",
    "WEBAPP_TIMEOUT": "60"
  }
}
<code>

FaceBlurTimer/function.json
<code>
{
  "scriptFile": "__init__.py",
  "bindings": [
    {
      "name": "mytimer",
      "type": "timerTrigger",
      "direction": "in",
      "schedule": "0 0 * * * *"
    }
  ]
}
```

Step 8: Set Application Settings

```

# Set your environment variables
az functionapp config appsettings set \
  --name myFaceBlurFunctionApp \
  --resource-group myResourceGroup \
  --settings \
  "AZURE_STORAGE_CONNECTION_STRING=your_blob_storage_connection_string" \
  "AZURE_CONTAINER_NAME=your_source_container_name" \
  "WEBAPP_URL=https://your-webapp.com/blur" \
  "WEBAPP_TIMEOUT=60"
```

Step 9: Initialize Function Project Locally

```
# Create project directory
mkdir face-blur-function
cd face-blur-function
# Initialize function project
```

```
func init --python
```

Step 10: Create Timer Function

```

func new --name FaceBlurTimer --template "Timer trigger"

<code>

Step 11: Deploy to Azure

<code>
# Deploy the function
func azure functionapp publish myFaceBlurFunctionApp
```

Alternative: Deploy using Azure CLI (Zip Deploy) If you prefer to deploy using zip:

```
Create deployment package
zip -r deployment.zip .
```

Deploy using az cli

```
az functionapp deployment source config-zip \
  --resource-group myResourceGroup \
  --name myFaceBlurFunctionApp \
  --src deployment.zip
```

function_app.py

```

import azure.functions as func
import datetime
import json
import logging

app = func.FunctionApp()
```

face_blur_processor.py

```


"""
Face Blur Processor Module for Azure Functions
"""

import os
import requests
import logging
import random
from datetime import datetime
from azure.storage.blob import BlobServiceClient
```

```
from azure.core.exceptions import AzureError

# Configure logging for Azure Functions
logger = logging.getLogger(__name__)

class FaceBlurProcessor:
    def __init__(self):
        # Azure Blob Storage configuration
        self.connection_string =
os.environ.get('AZURE_STORAGE_CONNECTION_STRING')
        self.container_name = os.environ.get('AZURE_CONTAINER_NAME',
'images')
        # Web app configuration
        webapp_url = os.environ.get('WEBAPP_URL',
'https://your-webapp.com/blur')
        # Auto-fix URLs missing scheme
        if not webapp_url.startswith(('http://', 'https://')):
            webapp_url = f'https://{webapp_url}'
        self.webapp_url = webapp_url
        self.webapp_timeout = int(os.environ.get('WEBAPP_TIMEOUT', '60'))
        # Initialize blob service client
        if not self.connection_string:
            raise ValueError("AZURE_STORAGE_CONNECTION_STRING environment
variable is required")
        self.blob_service_client =
BlobServiceClient.from_connection_string(self.connection_string)

    def ensure_container_exists(self):
        """Ensure the container exists, create it if it does not exist."""
        try:
            container_client =
self.blob_service_client.get_container_client(self.container_name)
            if not container_client.exists():
                logger.info(f"Container '{self.container_name}' does not
exist. Creating it...")
                container_client.create_container()
                logger.info(f"Created container: {self.container_name}")
            return True
        except AzureError as e:
            logger.error(f"Error ensuring container exists: {e}")
            return False
    def get_random_image(self):
        """Get a random image from the blob container."""
        try:
            # Ensure container exists first
            if not self.ensure_container_exists():
                return None, None
            container_client =
self.blob_service_client.get_container_client(self.container_name)
            # List all blobs
            blobs = list(container_client.list_blobs())
```

```
            if not blobs:
                logger.warning(f"No images found in container
'{self.container_name}'")
                logger.info("Please upload some images to the container
first")
                return None, None
            # Select a random blob
            random_blob = random.choice(blobs)
            logger.info(f"Randomly selected image: {random_blob.name}")
            logger.info(f"Total images available: {len(blobs)}")
            return random_blob.name, random_blob
        except AzureError as e:
            logger.error(f"Error accessing blob storage: {e}")
            return None, None
    def get_latest_image(self):
        """Get the most recent image from the blob container (kept for
backward compatibility)."""
        try:
            # Ensure container exists first
            if not self.ensure_container_exists():
                return None, None
            container_client =
self.blob_service_client.get_container_client(self.container_name)
            # List all blobs and find the most recent one
            blobs = list(container_client.list_blobs())
            if not blobs:
                logger.warning(f"No images found in container
'{self.container_name}'")
                logger.info("Please upload some images to the container
first")
                return None, None
            # Sort by last_modified to get the latest
            latest_blob = max(blobs, key=lambda x: x.last_modified)
            logger.info(f"Found latest image: {latest_blob.name}")
            return latest_blob.name, latest_blob
        except AzureError as e:
            logger.error(f"Error accessing blob storage: {e}")
            return None, None
    def call_blur_webapp(self, blob_name):
        """Call the web app to blur faces in the image by sending the blob
name."""
        try:
            # Send the blob name as form data (matching your webapp's
expectation)
            data = {'file': blob_name}
            logger.info(f"Making request to: {self.webapp_url} with blob
name: {blob_name}")
            response = requests.post(
                self.webapp_url,
                data=data,
                timeout=self.webapp_timeout
```

```
            )
            logger.info(f"Response status code: {response.status_code}")
            logger.info(f"Response headers: {dict(response.headers)}")
            if response.status_code == 200:
                logger.info("Successfully called face blur web app")
                try:
                    response_data = response.json()
                    logger.info(f"Response JSON: {response_data}")
                    blob_url = response_data.get('blob_url')
                    if blob_url:
                        logger.info(f"Received blob URL: {blob_url}")
                        return blob_url
                    else:
                        logger.error("No blob_url in response")
                        return None
                except ValueError as e:
                    logger.error(f"Response is not valid JSON: {e}")
                    logger.error(f"Response text: {response.text}")
                    return None
            else:
                logger.error(f"Web app returned status code:
{response.status_code}")
                logger.error(f"Response text: {response.text}")
                # Parse error response
                try:
                    error_data = response.json()
                    logger.error(f"Error details: {error_data}")
                    # Check for specific Azure errors
                    error_msg = error_data.get('error', '')
                    if 'OutOfRangeInput' in error_msg:
                        logger.error("Azure Face API OutOfRangeInput error -
image may not meet requirements")
                    elif 'InvalidImageFormat' in error_msg:
                        logger.error("Azure Face API InvalidImageFormat
error - image format not supported")
                    elif 'InvalidImageSize' in error_msg:
                        logger.error("Azure Face API InvalidImageSize error
- image size not supported")
                    elif 'NoFaceDetected' in error_msg:
                        logger.info("No faces detected in image - this might
be normal")
                        return None  # This might not be an error
                except ValueError:
                    logger.error("Error response is not JSON")
                return None
        except Exception as e:
            logger.error(f"Error calling web app: {e}")
            return None
    def log_processed_image(self, blob_url, original_blob_name):
        """Log the processed image URL (since your web app already uploads
to blob storage)."""
```

```
        try:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            # Log the processing completion
            logger.info(f"Processed image available at: {blob_url}")
            logger.info(f"Original image: {original_blob_name}")
            logger.info(f"Processing completed at: {timestamp}")
            return blob_url
        except Exception as e:
            logger.error(f"Error logging processed image: {e}")
            return None
    def process_random_image(self):
        """Main processing function - pull random image, blur faces, and log
result."""
        logger.info("Starting face blur processing (random image
selection)...")
        # Get random image
        blob_name, blob_info = self.get_random_image()
        if not blob_name:
            return False
        try:
            # Call web app to blur faces directly with blob name (no need to
download)
            blob_url = self.call_blur_webapp(blob_name)
            if not blob_url:
                return False
            # Log the processed image info (web app already uploaded it)
            result = self.log_processed_image(blob_url, blob_name)
            if result:
                logger.info(f"Successfully processed random image. Blurred
version at: {blob_url}")
                return True
            else:
                return False
        except Exception as e:
            logger.error(f"Unexpected error during processing: {e}")
            return False
    def process_latest_image(self):
        """Main processing function - pull latest image, blur faces, and log
result (kept for backward compatibility)."""
        logger.info("Starting face blur processing (latest image
selection)...")
        # Get latest image
        blob_name, blob_info = self.get_latest_image()
        if not blob_name:
            return False
        try:
            # Call web app to blur faces directly with blob name (no need to
download)
            blob_url = self.call_blur_webapp(blob_name)
            if not blob_url:
                return False
```

```
            # Log the processed image info (web app already uploaded it)
            result = self.log_processed_image(blob_url, blob_name)
            if result:
                logger.info(f"Successfully processed image. Blurred version
at: {blob_url}")
                return True
            else:
                return False
        except Exception as e:
            logger.error(f"Unexpected error during processing: {e}")
            return False
```

__init__.py

```

import datetime
import logging
import azure.functions as func
import sys
import os

# Add the parent directory to the path so we can import our modules
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from shared.face_blur_processor import FaceBlurProcessor


def main(mytimer: func.TimerRequest) -> None:
    """
    Azure Function that runs on a timer to process random images for face
blurring.
    """
    utc_timestamp = datetime.datetime.utcnow().replace(
        tzinfo=datetime.timezone.utc).isoformat()

    if mytimer.past_due:
        logging.info('The timer is past due!')

    logging.info('Face blur timer trigger function ran at %s',
utc_timestamp)
    try:
        # Initialize the face blur processor
        processor = FaceBlurProcessor()
        # Process a random image
        success = processor.process_random_image()
        if success:
            logging.info("Face blur processing completed successfully")
        else:
            logging.error("Face blur processing failed")
    except Exception as e:
        logging.error(f"Error in face blur function: {str(e)}")
```

```
        raise e

    logging.info('Face blur timer trigger function completed at %s',
datetime.datetime.utcnow().replace(tzinfo=datetime.timezone.utc).isoformat()
)
```