



CDW Documentation

AI Chatbot with Sentiment and In Kind Responses

AI Chatbot wit Sentiment and In Kind Responses

Requirements

1. Azure Resources Checklist

Resource Type	Name (per your setup)	Notes
Function App	dehamerfuncapp	Your C# Azure Function app
Storage Account	Linked to Function App	Required for deployment—should auto-create if <code>-functions-version</code> specified
Azure Cognitive Services (Text Analytics)	dehamersentimentai	For sentiment classification
Azure OpenAI	don-openai-useast	For GPT-4.1-based reply generation

Prerequisites

Install These

```
brew install dotnet-sdk #This will be version 9 which will not work but gives you the rest of the stuff needed.
brew install azure-cli
```

Assuming you already have created a Resource Group named `don-test-rg`, an OpenAI deployment named and deployed a gpt model named `gpt-4.1`.

Install this from the webpage and follow directions. [dotnet 8.x](#)

Then these

```
dotnet add package Microsoft.Azure.Functions.Worker.Extensions.OpenApi --version 1.4.0
dotnet --list-sdks
dotnet add package Azure.AI.TextAnalytics --version 5.3.0
dotnet add package Azure.AI.OpenAI --version 2.0.0
```

Create Storage Account

```
az storage account create \
  --name dehamerfuncstorage \
  --location eastus \
  --resource-group don-test-rg \
  --sku Standard_LRS
```

Create Function Plan for Windows

```
az functionapp plan create \  
  --resource-group don-test-rg \  
  --name donfuncplan \  
  --location eastus \  
  --sku B1 \  
  --is-linux false
```

Create Function App

```
az functionapp create \  
  --resource-group don-test-rg \  
  --name dehamerfuncapp \  
  --plan donfuncplan \  
  --storage-account dehamerfuncstorage \  
  --runtime dotnet-isolated \  
  --functions-version 4 \  
  --os-type Windows
```

Create TextAnalytics

```
az cognitiveservices account create \  
  --name dehamersentimentai \  
  --resource-group don-test-rg \  
  --location eastus \  
  --kind TextAnalytics \  
  --sku S \  
  --custom-domain dehamersentimentai  
  
az cognitiveservices account update \  
  --name donsmentimentai \  
  --resource-group don-test-rg \  
  --set properties.publicNetworkAccess=Enabled
```

Get keys and endpoint for export and app settings

```
export TEXT_ANALYTICS_ENDPOINT=`az cognitiveservices account show \  
  --name dehamersentimentai \  
  --resource-group don-test-rg \  
  --query "properties.endpoint"|sed -e s:\":::g`  
  
export TEXT_ANALYTICS_KEY=`az cognitiveservices account keys list \  
  --name dehamersentimentai \  
  --resource-group don-test-rg \  
  --query "key1" -o tsv`  
  
az functionapp config appsettings set \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --settings \  
  TEXT_ANALYTICS_ENDPOINT=$TEXT_ANALYTICS_ENDPOINT \  
  TEXT_ANALYTICS_KEY=$TEXT_ANALYTICS_KEY
```

```
TEXT_ANALYTICS_KEY=$TEXT_ANALYTICS_KEY
export OPENAI_KEY=`az cognitiveservices account keys list \
  --name don-openai-useast \
  --resource-group don-test-rg \
  --query "key1" -o tsv`

export OPENAI_ENDPOINT=https://don-openai-useast.openai.azure.com/

az functionapp config appsettings set \
  --name donfuncapp \
  --resource-group don-test-rg \
  --settings \
  OPENAI_KEY=$OPENAI_KEY \
  OPENAI_ENDPOINT=$OPENAI_ENDPOINT
#Confirm they were set
az functionapp config appsettings list \
  --name dehamerfuncapp \
  --resource-group don-test-rg \
  --query "[?starts_with(name, 'TEXT_') || starts_with(name, 'OPENAI_')]" -o
table
```

Create a directory to store the functionapp files

```
mkdir ~/FeedbackFunctionDotNet
cd ~/FeedbackFunctionDotNet
```

Save the FeedbackFunction.cs, FeedbackFunction.csproj, global.json, host.json, local.settings.json, and program.cs into the FeedBackFunctionDotNet directory.

[FeedbackFunction.cs](#)

```
using System.Net.Http.Headers;
using System.Text;
using System.Text.Json;
using Azure;
using Azure.AI.TextAnalytics;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Azure.Functions.Worker.Http;
using Microsoft.Extensions.Logging;

public class DeHamerFeedbackFunction
{
    private readonly ILogger _logger;

    public DeHamerFeedbackFunction(ILoggerFactory loggerFactory)
    {
        _logger =
loggerFactory.CreateLogger<DeHamerFeedbackFunction>();
    }

    [Function("DeHamerFeedbackFunction")]
```

```
public async Task<HttpresponseData> Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route =
null)]
    HttpRequestData req)
{
    _logger.LogInformation("DeHamerFeedbackFunction triggered.");

    var requestBody = await new
StreamReader(req.Body).ReadToEndAsync();
    var data = JsonSerializer.Deserialize<Dictionary<string,
string>>(requestBody);
    string feedback = data?["feedback"] ?? string.Empty;

    // Get Text Analytics config
    var textEndpoint =
Environment.GetEnvironmentVariable("TEXT_ANALYTICS_ENDPOINT");
    var textKey =
Environment.GetEnvironmentVariable("TEXT_ANALYTICS_KEY");

    _logger.LogInformation("Using Text Analytics endpoint:
{endpoint}", textEndpoint);

    var credentials = new AzureKeyCredential(textKey);
    var client = new TextAnalyticsClient(new Uri(textEndpoint),
credentials);
    var documentSentiment = await
client.AnalyzeSentimentAsync(feedback);
    var sentiment =
documentSentiment.Value.Sentiment.ToString().ToLower();

    _logger.LogInformation("Sentiment detected: {sentiment}",
sentiment);

    // Compose prompt
    string prompt = sentiment switch
    {
        "positive" => $"Respond in a cheerful and thankful tone to
the following positive customer feedback:\n\n{feedback}\n",
        "negative" => $"Respond in a harsh, sarcastic, and annoyed
tone to the following negative customer feedback:\n\n{feedback}\n",
        _ => $"Respond neutrally and professionally to the
following feedback:\n\n{feedback}\n"
    };

    // OpenAI Setup
    var openaiEndpoint =
Environment.GetEnvironmentVariable("OPENAI_API_ENDPOINT")?.TrimEnd('/')
;
    var openaiKey =
Environment.GetEnvironmentVariable("OPENAI_API_KEY");
    var deployment =
```

```
Environment.GetEnvironmentVariable("OPENAI_DEPLOYMENT");

_logger.LogInformation("Calling OpenAI at: {url}",
    $"{openaiEndpoint}/openai/deployments/{deployment}/chat/completions");

using var httpClient = new HttpClient();
httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", openaiKey);

var payload = JsonSerializer.Serialize(new
{
    messages = new[]
    {
        new { role = "system", content = "You are a customer
support agent who mirrors the customer's sentiment tone." },
        new { role = "user", content = prompt }
    }
});

_logger.LogInformation("Payload sent to OpenAI: {payload}",
    payload);

var response = await httpClient.PostAsync(
    $"{openaiEndpoint}/openai/deployments/{deployment}/chat/completions?api
-version=2024-02-15-preview",
    new StringContent(payload, Encoding.UTF8,
    "application/json"));

if (!response.IsSuccessStatusCode)
{
    var errorDetails = await
response.Content.ReadAsStringAsync();
    _logger.LogError("OpenAI API call failed: {status} -
{details}", response.StatusCode, errorDetails);

    var errorResponse =
req.CreateResponse(System.Net.HttpStatusCode.InternalServerError);
    await errorResponse.WriteAsJsonAsync(new { error = "Failed
to get OpenAI response", status = response.StatusCode, details =
errorDetails });
    return errorResponse;
}

var json = JsonDocument.Parse(await
response.Content.ReadAsStringAsync());
_logger.LogInformation("Raw OpenAI response: {json}", json);

var message =
json.RootElement.GetProperty("choices")[0].GetProperty("message").GetPr
operty("content").GetString();
```

```
    var result = new
    {
        sentiment = sentiment,
        message = message
    };

    var responseData =
req.CreateResponse(System.Net.HttpStatusCode.OK);
    await responseData.WriteAsJsonAsync(result);
    return responseData;
}
}
```

[FeedbackFunction.csproj](#)

```
<Project Sdk="Microsoft.NET.Sdk.Worker">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <AzureFunctionsVersion>v4</AzureFunctionsVersion>
    <OutputType>Exe</OutputType>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Azure.AI.TextAnalytics" Version="5.3.0"
/>
    <PackageReference Include="Microsoft.Azure.Functions.Worker"
Version="1.17.0" />
    <PackageReference
Include="Microsoft.Azure.Functions.Worker.Extensions.OpenApi"
Version="1.5.0" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.Sdk"
Version="1.17.0" OutputItemType="Analyzer" />
    <PackageReference
Include="Microsoft.Azure.Functions.Worker.Extensions.Http"
Version="3.1.0" />
    <PackageReference Include="Microsoft.Extensions.Logging.Console"
Version="8.0.0" />
    <PackageReference Include="System.Text.Json" Version="8.0.0" />
  </ItemGroup>

  <ItemGroup>
    <None Update="host.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
    <None Update="local.settings.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
      <CopyToPublishDirectory>Never</CopyToPublishDirectory>
    </None>
  </ItemGroup>
</Project>
```

```
</ItemGroup>  
  
</Project>
```

Make sure this matches the version of dotnet 8.x you install.

global.json

```
{  
  "sdk": {  
    "version": "8.0.411"  
  }  
}
```

host.json

```
{  
  "version": "2.0"  
}
```

local.settings.json

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

program.cs

```
using Microsoft.Extensions.Hosting;  
using Microsoft.Extensions.DependencyInjection;  
  
var host = new HostBuilder()  
    .ConfigureFunctionsWorkerDefaults()  
    .Build();  
  
host.Run();
```

From the FeedbackFunctionDonnet directory:

```
dotnet clean  
dotnet publish -c Release -o publish  
cd publish
```

```
zip -r ../publish.zip .
cd ..
az functionapp deployment source config-zip --resource-group don-test-rg --
name dehamerfuncapp --src publish.zip
```

After running you will see additional files in the FeedbackFunctionDotnet directory. This is normal.

```
ls
bin                FeedbackFunction.csproj host.json                obj
publish
FeedbackFunction.cs  global.json            local.settings.json
Program.cs          publish.zip
```

To test if it is working

```
export FUNCTION_KEY=`az functionapp function keys list \
  --name dehamerfuncapp \
  --resource-group don-test-rg \
  --function-name dehamerfeedbackfunction --query "default" | sed -e s:\":::`

curl -X POST
https://dehamerfuncapp.azurewebsites.net/api/dehamerfeedbackfunction \
  -H "x-functions-key: $FUNCTION_KEY" \
  -H "Content-Type: application/json" \
  -d '{"feedback": "This service was trash. I want my time back."}'
```

You should see something like:

```
{"sentiment":"negative","message":"Oh, absolutely\u2014because we totally
specialize in time travel, right? Sorry our \u0022trash\u0022 service
didn\u2019t meet your sky-high expectations. Your valuable time is obviously
worth so much, so thanks for investing it with us just to let us know how
you feel."}%
```

Confirmation tests

```
az functionapp function list --name dehamerfuncapp --resource-group don-
test-rg --query "[].{name:name, status:invokeUrlTemplate}" -o table
Name                Status
-----
dehamerfuncapp/DeHamerFeedbackFunction
https://dehamerfuncapp.azurewebsites.net/api/dehamerfeedbackfunction
```

```
az functionapp show --name dehamerfuncapp --resource-group don-test-rg --
query "enabledHostNames"
[
  "dehamerfuncapp.azurewebsites.net",
  "dehamerfuncapp.scm.azurewebsites.net"
]
```

```
az functionapp function list \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --query "[].invokeUrlTemplate" \  
  --output tsv
```

```
https://dehamerfuncapp.azurewebsites.net/api/dehamerfeedbackfunction
```

```
az functionapp function show \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --function-name dehamerfeedbackfunction \  
  --query "isDisabled"
```

```
false
```

If true

```
az functionapp function update \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --function-name dehamerfeedbackfunction \  
  --set isDisabled=false
```

Tailing logs if enabled. Notice it says webapp, not functionapp. This is because after deployment the functionapp is in a similar deployment mode to webapps and this command tails logs for both types. You would run this in one terminal logged in with az login and then run the curl from another terminal, also logged in unless you already have the key in your curl statement.

```
az webapp log tail \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg
```

If you want it to autostart after issues.

```
#Optional  
az webapp config set \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --always-on true
```

Get a list of your functionapps

```
az functionapp function list \  
  --name dehamerfuncapp \  
  --resource-group don-test-rg \  
  --output table
```

```
Href  
InvokeUrlTemplate
```

IsDisabled	Language	Location	Name
ResourceGroup	ScriptHref		
TestData	TestDataHref		
-----	-----	-----	-----
-----	-----	-----	-----
-----	-----	-----	-----
-----	-----	-----	-----
-----	-----	-----	-----
-----	-----	-----	-----
https://dehamerfuncapp.azurewebsites.net/admin/functions/DeHamerFeedbackFunction	https://dehamerfuncapp.azurewebsites.net/api/dehamerfeedbackfunction		
False	dotnet-isolated	East US	
dehamerfuncapp/DeHamerFeedbackFunction		don-test-rg	
https://dehamerfuncapp.azurewebsites.net/admin/vfs/site/wwwroot/FeedbackFunction.dll			
https://dehamerfuncapp.azurewebsites.net/admin/vfs/data/Functions/sampledData/DeHamerFeedbackFunction.dat			

Only do this for testing. It's not secure in the real world unless you don't want to have domain restrictions on access to the functionapp. I had to do it so I could test from my home webserver.

```
az functionapp cors add \
  --name dehamerfuncapp \
  --resource-group don-test-rg \
  --allowed-origins "*"

```

[AI Knowledge](#)