

# Document Classification Project - Complete Setup Guide for VSCode

## Prerequisites Checklist

### Install These Before Starting:

#### 1. Visual Studio Code

- Download: <https://code.visualstudio.com/>

#### 2. .NET 8 SDK

- Download: <https://dotnet.microsoft.com/download/dotnet/8.0>
- Verify installation: `(dotnet --version)`

#### 3. Azure Functions Core Tools v4

```
bash
```

```
npm install -g azure-functions-core-tools@4 --unsafe-perm true
```

- Verify: `(func --version)`

#### 4. Azure CLI

- Download: <https://docs.microsoft.com/cli/azure/install-azure-cli>
- Verify: `(az --version)`

#### 5. Node.js & NPM (for Azurite)

- Download: <https://nodejs.org/>
- Verify: `(node --version)`

#### 6. Git

- Download: <https://git-scm.com/>
- Verify: `(git --version)`

---

## VSCode Extensions to Install

Open VSCode and install these extensions:

1. **Azure Functions** (ms-azuretools.vscode-azurefunctions)
2. **Azure Account** (ms-vscode.azure-account)
3. **C# Dev Kit** (ms-dotnettools.csdevkit)

4. **Azure Resources** (ms-azuretools.vscode-azureresourcegroups)

5. **REST Client** (humao.rest-client)

6. **Azure Storage** (ms-azuretools.vscode-azurestorage)

### How to install:

- Press `Ctrl+Shift+X` (Windows/Linux) or `Cmd+Shift+X` (Mac)
  - Search for each extension name
  - Click "Install"
- 

## 📁 STEP 1: Create Project Folder Structure

Open Terminal in VSCode (`Ctrl+~` or `Cmd+~`) and run:

```
bash

# Create main project folder
mkdir DocumentClassificationProject
cd DocumentClassificationProject

# Create subfolders
mkdir AzureFunctions
mkdir WebApp
mkdir Scripts
mkdir Documentation

# Navigate to Functions folder
cd AzureFunctions
```

## ⚙️ STEP 2: Initialize Azure Functions Project

In the `AzureFunctions` folder:

```
bash

# Initialize the project
func init DocumentClassification --worker-runtime dotnet-isolated --target-framework net8.0

# Navigate into the project
cd DocumentClassification
```

## STEP 3: Create Project Files

### A. Create `DocumentClassification.csproj`

Create/replace the file with this content:

```
xml

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

  <ItemGroup>
    <PackageReference Include="Microsoft.Azure.Functions.Worker" Version="1.21.0" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.Sdk" Version="1.16.4" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.Extensions.DurableTask" Version="1.1.0" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.Extensions.ServiceBus" Version="5.16.0" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.Extensions.Http" Version="3.1.0" />
    <PackageReference Include="Azure.AI.FormRecognizer" Version="4.1.0" />
    <PackageReference Include="Azure.Search.Documents" Version="11.5.1" />
    <PackageReference Include="Microsoft.Azure.Cosmos" Version="3.38.1" />
    <PackageReference Include="Azure.AI.OpenAI" Version="1.0.0-beta.14" />
    <PackageReference Include="Microsoft.SemanticKernel" Version="1.0.1" />
    <PackageReference Include="Microsoft.ApplicationInsights.WorkerService" Version="2.21.0" />
    <PackageReference Include="Microsoft.Azure.Functions.Worker.ApplicationInsights" Version="1.1.0" />
  </ItemGroup>

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

### B. Restore NuGet Packages

```
bash
```

```
dotnet restore
```

## 🔑 STEP 4: Azure Login & Resource Setup

### Login to Azure

```
bash
```

```
# Login to Azure
```

```
az login
```

```
# List your subscriptions
```

```
az account list --output table
```

```
# Set your subscription (replace with your subscription ID)
```

```
az account set --subscription "YOUR-SUBSCRIPTION-ID"
```

### Create Azure Resources Script

Create file: [../../Scripts/create-azure-resources.sh](#)

```
bash
```

```

#!/bin/bash

# =====
# Azure Document Classification Setup Script
# =====

# Configuration
RESOURCE_GROUP="rg-doc-classification"
LOCATION="eastus"
TIMESTAMP=$(date +%s)
STORAGE_ACCOUNT="stdocclass${TIMESTAMP}"
SERVICE_BUS="sbdocclass${TIMESTAMP}"
COSMOS_DB="cosmosdocclass${TIMESTAMP}"
DOC_INTELLIGENCE="docintel${TIMESTAMP}"
OPENAI_SERVICE="openaidoc${TIMESTAMP}"
SEARCH_SERVICE="searchdoc${TIMESTAMP}"

echo "====="
echo "Creating Azure Resources for Document Classification"
echo "====="
echo ""

# Create Resource Group
echo "📦 Creating Resource Group: $RESOURCE_GROUP"
az group create --name $RESOURCE_GROUP --location $LOCATION
echo "✓ Resource Group created"
echo ""

# Create Storage Account
echo "💾 Creating Storage Account: $STORAGE_ACCOUNT"
az storage account create \
--name $STORAGE_ACCOUNT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--sku Standard_LRS \
--kind StorageV2
echo "✓ Storage Account created"

# Get Storage Key
STORAGE_KEY=$(az storage account keys list \
--account-name $STORAGE_ACCOUNT \
--resource-group $RESOURCE_GROUP \
--query "[0].value" -o tsv)

# Create Blob Container
echo "📁 Creating Blob Container: documents"

```

```

az storage container create \
--name documents \
--account-name $STORAGE_ACCOUNT \
--account-key $STORAGE_KEY
echo "✅ Container created"
echo ""

# Create Service Bus
echo "🚂 Creating Service Bus: $SERVICE_BUS"
az servicebus namespace create \
--name $SERVICE_BUS \
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--sku Standard

az servicebus queue create \
--name document-processing-queue \
--namespace-name $SERVICE_BUS \
--resource-group $RESOURCE_GROUP
echo "✅ Service Bus created"
echo ""

# Create Cosmos DB
echo "🌐 Creating Cosmos DB: $COSMOS_DB"
az cosmosdb create \
--name $COSMOS_DB \
--resource-group $RESOURCE_GROUP \
--locations regionName=$LOCATION \
--default-consistency-level Session

az cosmosdb sql database create \
--account-name $COSMOS_DB \
--resource-group $RESOURCE_GROUP \
--name DocumentMetadata

az cosmosdb sql container create \
--account-name $COSMOS_DB \
--database-name DocumentMetadata \
--name Documents \
--partition-key-path "/documentId" \
--resource-group $RESOURCE_GROUP \
--throughput 400
echo "✅ Cosmos DB created"
echo ""

# Create Document Intelligence
echo "📄 Creating Document Intelligence: $DOC_INTELLIGENCE"

```

```
az cognitiveservices account create \
--name $DOC_INTELLIGENCE \
--resource-group $RESOURCE_GROUP \
--kind FormRecognizer \
--sku S0 \
--location $LOCATION \
--yes
echo "✅ Document Intelligence created"
echo ""
```

*# Create Azure OpenAI*

```
echo "🤖 Creating Azure OpenAI: $OPENAI_SERVICE"
az cognitiveservices account create \
--name $OPENAI_SERVICE \
--resource-group $RESOURCE_GROUP \
--kind OpenAI \
--sku S0 \
--location $LOCATION \
--yes
```

*# Deploy embedding model*

```
echo "📊 Deploying embedding model..."
az cognitiveservices account deployment create \
--name $OPENAI_SERVICE \
--resource-group $RESOURCE_GROUP \
--deployment-name text-embedding-ada-002 \
--model-name text-embedding-ada-002 \
--model-version "2" \
--model-format OpenAI \
--sku-name "Standard" \
--sku-capacity 1
echo "✅ Azure OpenAI created and model deployed"
echo ""
```

*# Create AI Search*

```
echo "🔍 Creating AI Search: $SEARCH_SERVICE"
az search service create \
--name $SEARCH_SERVICE \
--resource-group $RESOURCE_GROUP \
--sku basic \
--location $LOCATION
echo "✅ AI Search created"
echo ""
```

```
# =====
```

*# Retrieve Connection Strings*

```
# =====
```

```
echo ""
echo "====="
echo "📋 CONNECTION STRINGS & KEYS"
echo "====="
echo ""

echo "🔒 STORAGE CONNECTION STRING:"
STORAGE_CONN=$(az storage account show-connection-string \
--name $STORAGE_ACCOUNT \
--resource-group $RESOURCE_GROUP \
--query connectionString -o tsv)
echo "$STORAGE_CONN"
echo ""

echo "🔒 SERVICE BUS CONNECTION STRING:"
SERVICEBUS_CONN=$(az servicebus namespace authorization-rule keys list \
--namespace-name $SERVICE_BUS \
--name RootManageSharedAccessKey \
--resource-group $RESOURCE_GROUP \
--query primaryConnectionString -o tsv)
echo "$SERVICEBUS_CONN"
echo ""

echo "🔒 COSMOS DB CONNECTION STRING:"
COSMOS_CONN=$(az cosmosdb keys list \
--name $COSMOS_DB \
--resource-group $RESOURCE_GROUP \
--type connection-strings \
--query "connectionStrings[0].connectionString" -o tsv)
echo "$COSMOS_CONN"
echo ""

echo "🔒 DOCUMENT INTELLIGENCE:"
DOC_INTEL_ENDPOINT="https://$LOCATION.api.cognitive.microsoft.com/"
DOC_INTEL_KEY=$(az cognitiveservices account keys list \
--name $DOC_INTELLIGENCE \
--resource-group $RESOURCE_GROUP \
--query key1 -o tsv)
echo "Endpoint: $DOC_INTEL_ENDPOINT"
echo "Key: $DOC_INTEL_KEY"
echo ""

echo "🔒 AZURE OPENAI:"
OPENAI_ENDPOINT="https://$OPENAI_SERVICE.openai.azure.com/"
OPENAI_KEY=$(az cognitiveservices account keys list \
--name $OPENAI_SERVICE \
```

```

--resource-group $RESOURCE_GROUP \
--query key1 -o tsv)
echo "Endpoint: $OPENAI_ENDPOINT"
echo "Key: $OPENAI_KEY"
echo ""

echo "🔒 AZURE AI SEARCH:"
SEARCH_ENDPOINT="https://$SEARCH_SERVICE.search.windows.net"
SEARCH_KEY=$(az search admin-key show \
--service-name $SEARCH_SERVICE \
--resource-group $RESOURCE_GROUP \
--query primaryKey -o tsv)
echo "Endpoint: $SEARCH_ENDPOINT"
echo "Key: $SEARCH_KEY"
echo ""

# =====
# Generate local.settings.json
# =====

echo "====="
echo "📝 Generating local.settings.json"
echo "====="

cat > ../../AzureFunctions/DocumentClassification/local.settings.json << EOF
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "UseDevelopmentStorage=true",
        "FUNCTIONS_WORKER_RUNTIME": "dotnet-isolated",

        "StorageConnection": "$STORAGE_CONN",
        "ServiceBusConnection": "$SERVICEBUS_CONN",
        "CosmosDBConnection": "$COSMOS_CONN",

        "DocumentIntelligenceEndpoint": "$DOC_INTEL_ENDPOINT",
        "DocumentIntelligenceKey": "$DOC_INTEL_KEY",

        "OpenAIEndpoint": "$OPENAI_ENDPOINT",
        "OpenAIKey": "$OPENAI_KEY",
        "OpenAIEmbeddingModel": "text-embedding-ada-002",

        "SearchEndpoint": "$SEARCH_ENDPOINT",
        "SearchKey": "$SEARCH_KEY",
        "SearchIndexName": "documents-index"
    }
}

```

```
EOF
```

```
echo "✅ local.settings.json created successfully!"  
echo ""  
  
echo "===== "  
echo "🎉 Setup Complete!"  
echo "===== "  
echo ""  
echo "Next Steps:"  
echo "1. Navigate to: AzureFunctions/DocumentClassification"  
echo "2. Run: func start"  
echo "3. Test your functions!"  
echo ""  
echo "Resource Group: $RESOURCE_GROUP"  
echo "Location: $LOCATION"  
echo "===== "
```

## Make Script Executable & Run It

```
bash  
  
# Make script executable  
chmod +x ../../Scripts/create-azure-resources.sh  
  
# Run the script  
../../Scripts/create-azure-resources.sh
```

⚠️ **IMPORTANT:** Save all the connection strings output by the script!

## STEP 5: Create Function Code Files

Navigate back to your Functions project folder:

```
bash  
  
cd ~/DocumentClassificationProject/AzureFunctions/DocumentClassification
```

### File 1: **Models.cs**

```
csharp
```

```
namespace DocumentClassification;

public class DocumentInfo
{
    public string BlobUrl { get; set; } = string.Empty;
    public string DocumentId { get; set; } = string.Empty;
    public string FileName { get; set; } = string.Empty;
}

public class EmbeddedDocument
{
    public string DocumentId { get; set; } = string.Empty;
    public string DocumentType { get; set; } = string.Empty;
    public string Content { get; set; } = string.Empty;
    public int StartPage { get; set; }
    public int EndPage { get; set; }
    public string CorrelationId { get; set; } = Guid.NewGuid().ToString();
}

public class DocumentMetadata
{
    public string id { get; set; } = string.Empty;
    public string documentId { get; set; } = string.Empty;
    public string documentType { get; set; } = string.Empty;
    public int startPage { get; set; }
    public int endPage { get; set; }
    public DateTime timestamp { get; set; }
}
```

## File 2: Program.cs

csharp

```
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var host = new HostBuilder()
    .ConfigureFunctionsWebApplication()
    .ConfigureServices(services =>
{
    services.AddApplicationInsightsTelemetryWorkerService();
    services.ConfigureFunctionsApplicationInsights();
})
.Build();

host.Run();
```

**File 3: DocumentOrchestrator.cs**

csharp

```
using Microsoft.Azure.Functions.Worker;
using Microsoft.Azure.Functions.Worker.Http;
using Microsoft.DurableTask;
using Microsoft.DurableTask.Client;
using Microsoft.Extensions.Logging;
using System.Text.Json;

namespace DocumentClassification;

public class DocumentOrchestrator
{
    [Function(nameof(DocumentOrchestrator))]
    public static async Task<List<string>> RunOrchestrator(
        [OrchestrationTrigger] TaskOrchestrationContext context)
    {
        ILogger logger = context.CreateReplaySafeLogger(nameof(DocumentOrchestrator));

        var documentInfo = context.GetInput<DocumentInfo>();
        logger.LogInformation($"🚀 Starting orchestration for: {documentInfo?.FileName}");

        // Step 1: Analyze document
        logger.LogInformation("📝 Step 1: Analyzing document...");

        var analyzedDocs = await context.CallActivityAsync<List<EmbeddedDocument>>(
            nameof(AnalyzeDocumentActivity),
            documentInfo);

        logger.LogInformation($"✅ Found {analyzedDocs.Count} embedded documents");

        var results = new List<string>();

        // Step 2: Process each document in parallel
        logger.LogInformation("⚙️ Step 2: Processing documents...");

        var metadataTasks = new List<Task>();
        var embeddingTasks = new List<Task>();

        foreach (var doc in analyzedDocs)
        {
            // Store metadata
            metadataTasks.Add(context.CallActivityAsync(
                nameof(StoreMetadataActivity),
                doc));
        }

        // Create embeddings
        embeddingTasks.Add(context.CallActivityAsync(
            nameof(CreateEmbeddingsActivity),
```

```
        doc));  
  
        results.Add($"✓ Processed: {doc.DocumentType} ({Pages} {doc.StartPage}-{doc.EndPage})");  
    }  
  
    await Task.WhenAll(metadataTasks);  
    logger.LogInformation("✓ All metadata stored");  
  
    await Task.WhenAll(embeddingTasks);  
    logger.LogInformation("✓ All embeddings created");  
  
    logger.LogInformation($"🎉 Orchestration complete! Processed {analyzedDocs.Count} documents");  
  
    return results;  
}  
  
[Function(nameof(HttpStart))]  
public static async Task<HttpResponseData> HttpStart(  
    [HttpTrigger(AuthorizationLevel.Anonymous, "post")] HttpRequestData req,  
    [DurableClient] DurableTaskClient client,  
    FunctionContext executionContext)  
{  
    ILogger logger = executionContext.GetLogger(nameof(HttpStart));  
  
    var documentInfo = await req.ReadFromJsonAsync<DocumentInfo>();  
  
    string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(  
        nameof(DocumentOrchestrator),  
        documentInfo);  
  
    logger.LogInformation($"✓ Started orchestration with ID = '{instanceId}'");  
  
    return client.CreateCheckStatusResponse(req, instanceId);  
}  
  
[Function(nameof(ServiceBusStart))]  
public static async Task ServiceBusStart(  
    [ServiceBusTrigger("document-processing-queue", Connection = "ServiceBusConnection")]  
    string message,  
    [DurableClient] DurableTaskClient client,  
    FunctionContext executionContext)  
{  
    ILogger logger = executionContext.GetLogger(nameof(ServiceBusStart));  
  
    var documentInfo = JsonSerializer.Deserialize<DocumentInfo>(message);  
  
    string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(
```

```
        nameof(DocumentOrchestrator),  
        documentInfo);  
  
    logger.LogInformation($" Started orchestration from Service Bus. ID = '{instanceId}'");  
}
```

#### File 4: **AnalyzeDocumentActivity.cs**

csharp

```
using Azure;
using Azure.AI.FormRecognizer.DocumentAnalysis;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;

namespace DocumentClassification;

public class AnalyzeDocumentActivity
{
    private readonly ILogger<AnalyzeDocumentActivity> _logger;

    public AnalyzeDocumentActivity(ILogger<AnalyzeDocumentActivity> logger)
    {
        _logger = logger;
    }

    [Function(nameof(AnalyzeDocumentActivity))]
    public async Task<List<EmbeddedDocument>> Run(
        [ActivityTrigger] DocumentInfo documentInfo)
    {
        _logger.LogInformation($"📝 Analyzing document: {documentInfo.BlobUrl}");

        var endpoint = Environment.GetEnvironmentVariable("DocumentIntelligenceEndpoint");
        var apiKey = Environment.GetEnvironmentVariable("DocumentIntelligenceKey");

        if (string.IsNullOrEmpty(endpoint) || string.IsNullOrEmpty(apiKey))
        {
            throw new InvalidOperationException("Document Intelligence credentials not configured");
        }

        var client = new DocumentAnalysisClient(
            new Uri(endpoint),
            new AzureKeyCredential(apiKey));

        try
        {
            var operation = await client.AnalyzeDocumentFromUriAsync(
                WaitUntil.Completed,
                "prebuilt-layout",
                new Uri(documentInfo.BlobUrl));

            var result = operation.Value;
            var embeddedDocs = new List<EmbeddedDocument>();

            // Extract content and page information
            if (result.Pages.Count > 0)
```

```

{
    embeddedDocs.Add(new EmbeddedDocument
    {
        DocumentId = documentInfo.DocumentId,
        DocumentType = "PDF", // You can enhance this to detect actual type
        Content = result.Content,
        StartPage = 1,
        EndPage = result.Pages.Count,
        CorrelationId = Guid.NewGuid().ToString()
    });
}

_logger.LogInformation($"✅ Found {embeddedDocs.Count} embedded documents");
return embeddedDocs;
}
catch (Exception ex)
{
    _logger.LogError($"❌ Error analyzing document: {ex.Message}");
    throw;
}
}
}
}

```

#### File 5: **StoreMetadataActivity.cs**

csharp

```
using Microsoft.Azure.Cosmos;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;

namespace DocumentClassification;

public class StoreMetadataActivity
{
    private readonly ILogger<StoreMetadataActivity> _logger;
    private static CosmosClient? _cosmosClient;

    public StoreMetadataActivity(ILogger<StoreMetadataActivity> logger)
    {
        _logger = logger;
    }

    [Function(nameof(StoreMetadataActivity))]
    public async Task Run([ActivityTrigger] EmbeddedDocument document)
    {
        _logger.LogInformation($"💾 Storing metadata for: {document.CorrelationId}");

        var connectionString = Environment.GetEnvironmentVariable("CosmosDBConnection");

        if (string.IsNullOrEmpty(connectionString))
        {
            throw new InvalidOperationException("Cosmos DB connection string not configured");
        }

        try
        {
            _cosmosClient ??= new CosmosClient(connectionString);

            var container = _cosmosClient.GetContainer("DocumentMetadata", "Documents");

            var metadata = new DocumentMetadata
            {
                id = document.CorrelationId,
                documentId = document.DocumentId,
                documentType = document.DocumentType,
                startPage = document.StartPage,
                endPage = document.EndPage,
                timestamp = DateTime.UtcNow
            };

            await container.CreateItemAsync(metadata, new PartitionKey(document.DocumentId));
        }
    }
}
```

```
_logger.LogInformation($"✓ Metadata stored successfully for {document.CorrelationId}");  
}  
catch (Exception ex)  
{  
    _logger.LogError($"✗ Error storing metadata: {ex.Message}");  
    throw;  
}  
}  
}
```

#### File 6: [CreateEmbeddingsActivity.cs](#)

csharp

```
using Azure;
using Azure.AI.OpenAI;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;

namespace DocumentClassification;

public class CreateEmbeddingsActivity
{
    private readonly ILogger<CreateEmbeddingsActivity> _logger;

    public CreateEmbeddingsActivity(ILogger<CreateEmbeddingsActivity> logger)
    {
        _logger = logger;
    }

    [Function(nameof(CreateEmbeddingsActivity))]
    public async Task Run([ActivityTrigger] EmbeddedDocument document)
    {
        _logger.LogInformation($"🤖 Creating embeddings for: {document.CorrelationId}");

        var endpoint = Environment.GetEnvironmentVariable("OpenAIEndpoint");
        var apiKey = Environment.GetEnvironmentVariable("OpenAIKey");
        var model = Environment.GetEnvironmentVariable("OpenAIEmbeddingModel");

        if (string.IsNullOrEmpty(endpoint) || string.IsNullOrEmpty(apiKey) || string.IsNullOrEmpty(model))
        {
            throw new InvalidOperationException("OpenAI credentials not configured");
        }

        try
        {
            var client = new OpenAIClient(new Uri(endpoint), new AzureKeyCredential(apiKey));

            // Simple chunking - take first 8000 chars for demo
            var contentChunk = document.Content.Length > 8000
                ? document.Content.Substring(0, 8000)
                : document.Content;

            var embeddingResponse = await client.GetEmbeddingsAsync(
                new EmbeddingsOptions(model, new[] { contentChunk }));

            var embedding = embeddingResponse.Value.Data[0].Embedding.ToArray();

            _logger.LogInformation($"✅ Created embedding with {embedding.Length} dimensions");
        }
    }
}
```

```
// TODO: In Phase 2, we'll add Azure AI Search indexing here

}

catch (Exception ex)
{
    _logger.LogError($"X Error creating embeddings: {ex.Message}");
    throw;
}

}

}
```

## File 7: host.json

```
json
{
  "version": "2.0",
  "logging": {
    "applicationInsights": {
      "samplingSettings": {
        "isEnabled": true,
        "maxTelemetryItemsPerSecond": 20
      }
    },
    "logLevel": {
      "default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "extensions": {
    "durableTask": {
      "hubName": "DocumentClassificationHub",
      "storageProvider": {
        "connectionStringName": "AzureWebJobsStorage"
      }
    }
  }
}
```



## STEP 6: Install Azurite (Local Storage Emulator)

Open a **NEW terminal** in VSCode:

bash

```
# Install Azurite globally  
npm install -g azurite  
  
# Create folder for Azurite data  
mkdir ~/azurite-data  
  
# Start Azurite  
azurite --silent --location ~/azurite-data --debug ~/azurite-data/debug.log
```

**Keep this terminal running!**

---

## ▶ STEP 7: Run Your Functions Locally

Open **another new terminal** in VSCode:

```
bash  
  
# Navigate to your project  
cd ~/DocumentClassificationProject/AzureFunctions/DocumentClassification  
  
# Build the project  
dotnet build  
  
# Start Functions  
func start
```

You should see output like:

Functions:

DocumentOrchestrator: orchestrationTrigger

HttpStart: [POST] http://localhost:7071/api/HttpStart

ServiceBusStart: serviceBusTrigger

AnalyzeDocumentActivity: activityTrigger

CreateEmbeddingsActivity: activityTrigger

StoreMetadataActivity: activityTrigger

## STEP 8: Test Your Functions

### Create Test File

Create file: `test-request.http`

```
http  
### Test Document Upload  
POST http://localhost:7071/api/HttpStart  
Content-Type: application/json  
  
{  
  "blobUrl": "https://YOUR_STORAGE_ACCOUNT.blob.core.windows.net/documents/sample.pdf",  
  "documentId": "test-001",  
  "fileName": "sample.pdf"  
}  
  
### Check Status (replace {instanceId} with actual ID from response)  
GET http://localhost:7071/runtime/webhooks/durabletask/instances/{instanceId}
```

### Test Using cURL

```
bash  
  
curl -X POST http://localhost:7071/api/HttpStart \  
-H "Content-Type: application/json" \  
-d '{  
  "blobUrl": "https://YOUR_STORAGE.blob.core.windows.net/documents/test.pdf",  
  "documentId": "test-001",  
  "fileName": "test.pdf"  
}'
```

## STEP 9: Deploy to Azure (When Ready)

```
bash
```

```
# Create Function App in Azure
az functionapp create \
--name func-doc-class-$(date +%s) \
--resource-group rg-doc-classification \
--consumption-plan-location eastus \
--runtime dotnet-isolated \
--functions-version 4 \
--storage-account YOUR_STORAGE_ACCOUNT_NAME

# Deploy
func azure functionapp publish YOUR_FUNCTION_APP_NAME
```

## Troubleshooting

**Issue:** "Cannot find module 'azurite'"

**Solution:** Install Azurite globally

```
bash
npm install -g azurite
```

**Issue:** "Storage emulator not running"

**Solution:** Start Azurite in a separate terminal

```
bash
azurite --silent --location ~/azurite-data
```

**Issue:** "Connection string invalid"

**Solution:** Check your `local.settings.json` file has correct connection strings from the setup script

**Issue:** Functions not starting

**Solution:**

```
bash
dotnet clean
dotnet build
func start --verbose
```

## Next Steps

1.  Test locally with sample PDF
  2.  Verify data in Cosmos DB
  3.  Check embeddings are created
  4.  Deploy to Azure
  5.  Create simple upload web app
- 

## Quick Command Reference

```
bash

# Start Azurite
azurite --silent --location ~/azurite-data

# Run Functions
cd ~/DocumentClassificationProject/AzureFunctions/DocumentClassification
func start

# Build project
dotnet build

# Clean build
dotnet clean && dotnet build

# View logs
func start --verbose

# Deploy to Azure
func azure functionapp publish YOUR_APP_NAME
```

## Need Help?

If you encounter issues:

1. Check Azurite is running
2. Verify `local.settings.json` has correct values
3. Run `dotnet build` to check for errors
4. Check Azure Portal for resource status

- 
5. Use `func start --verbose` for detailed logs

---

 **You're all set! Start with running the Azure setup script, then build and test your functions locally.**