

**this will be deployed to a windows server -- specify how we will host the fastapi service via IIS and how we will host the inference service using NSSM (following best practices for python services on windows)**

**Addendum C: Windows Server Deployment Architecture**

**IIS FastAPI Hosting & NSSM Service Management**

**Addendum Date**: September 10, 2025  
**Document Version**: 1.3

**Windows Server Deployment Overview**

The Baker Group LLM solution implements a **Windows Server-native deployment architecture** utilizing IIS for FastAPI API service hosting and NSSM (Non-Sucking Service Manager) for robust inference service management. This approach leverages Windows Server's enterprise-grade service management capabilities while maintaining optimal performance for GPU-accelerated AI workloads.

**Deployment Architecture Components**

**IIS-Hosted API Service**

* FastAPI application served through IIS with reverse proxy configuration
* Integrated Windows Authentication and SSL termination
* Application pool isolation and automatic restart capabilities
* Performance monitoring and logging integration

**NSSM-Managed Inference Service**

* Python inference service running as Windows Service via NSSM
* Automatic startup, restart, and failure recovery
* GPU resource management and process monitoring
* Isolated service execution with dedicated user context

**IIS FastAPI Configuration**

**Application Pool Setup**

<!-- applicationHost.config -->  
<system.applicationHost>  
 <applicationPools>  
 <add name="BakerGroupLLMAPI"   
 managedRuntimeVersion=""   
 processModel.identityType="ApplicationPoolIdentity"  
 processModel.idleTimeout="00:00:00"  
 processModel.maxProcesses="1"  
 recycling.periodicRestart.time="00:00:00">  
 <environmentVariables>  
 <add name="PYTHONPATH" value="C:\BakerGroup\LLM\api" />  
 <add name="INFERENCE\_SERVICE\_URL" value="http://localhost:8001" />  
 <add name="DATABASE\_PATH" value="C:\BakerGroup\LLM\data\queue.db" />  
 <add name="LOG\_LEVEL" value="INFO" />  
 </environmentVariables>  
 </add>  
 </applicationPools>  
</system.applicationHost>

**Web.config for FastAPI Reverse Proxy**

<?xml version="1.0" encoding="utf-8"?>  
<configuration>  
 <system.webServer>  
 <handlers>  
 <add name="httpPlatformHandler"   
 path="\*"   
 verb="\*"   
 modules="httpPlatformHandler"   
 resourceType="Unspecified" />  
 </handlers>  
 <httpPlatform processPath="C:\BakerGroup\LLM\venv\Scripts\python.exe"  
 arguments="C:\BakerGroup\LLM\api\main.py"  
 stdoutLogEnabled="true"  
 stdoutLogFile="C:\BakerGroup\LLM\logs\api-stdout.log"  
 startupTimeLimit="60"  
 requestTimeout="00:04:00">  
 <environmentVariables>  
 <environmentVariable name="PORT" value="%HTTP\_PLATFORM\_PORT%" />  
 <environmentVariable name="PYTHONPATH" value="C:\BakerGroup\LLM\api" />  
 </environmentVariables>  
 </httpPlatform>  
   
 <security>  
 <requestFiltering>  
 <requestLimits maxAllowedContentLength="104857600" /> <!-- 100MB -->  
 </requestFiltering>  
 </security>  
   
 <defaultDocument>  
 <files>  
 <clear />  
 </files>  
 </defaultDocument>  
   
 <staticContent>  
 <mimeMap fileExtension=".json" mimeType="application/json" />  
 </staticContent>  
 </system.webServer>  
   
 <system.web>  
 <compilation tempDirectory="C:\BakerGroup\LLM\temp\" />  
 </system.web>  
</configuration>

**FastAPI Application Entry Point**

# C:\BakerGroup\LLM\api\main.py  
import os  
import sys  
import uvicorn  
from pathlib import Path  
  
# Add project root to path  
project\_root = Path(\_\_file\_\_).parent  
sys.path.insert(0, str(project\_root))  
  
from api\_service import APIService  
  
def create\_app():  
 """Create FastAPI application for IIS hosting"""  
 api\_service = APIService()  
   
 # Configure for Windows/IIS deployment  
 api\_service.app.title = "Baker Group LLM API"  
 api\_service.app.version = "1.0.0"  
   
 # Add Windows-specific middleware  
 @api\_service.app.middleware("http")  
 async def add\_process\_time\_header(request, call\_next):  
 import time  
 start\_time = time.time()  
 response = await call\_next(request)  
 process\_time = time.time() - start\_time  
 response.headers["X-Process-Time"] = str(process\_time)  
 return response  
   
 return api\_service.app  
  
app = create\_app()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Get port from IIS HTTP Platform Handler  
 port = int(os.environ.get("PORT", 8000))  
   
 uvicorn.run(  
 app,  
 host="127.0.0.1",  
 port=port,  
 log\_level="info",  
 access\_log=True,  
 loop="asyncio"  
 )

**NSSM Inference Service Configuration**

**NSSM Service Installation Script**

@echo off  
REM install\_inference\_service.bat  
REM Run as Administrator  
  
cd /d "C:\BakerGroup\LLM"  
  
REM Download and install NSSM if not present  
if not exist "nssm.exe" (  
 echo Downloading NSSM...  
 powershell -Command "Invoke-WebRequest -Uri 'https://nssm.cc/release/nssm-2.24.zip' -OutFile 'nssm.zip'"  
 powershell -Command "Expand-Archive -Path 'nssm.zip' -DestinationPath '.'"  
 copy "nssm-2.24\win64\nssm.exe" .  
 rmdir /s /q "nssm-2.24"  
 del "nssm.zip"  
)  
  
REM Install the service  
nssm.exe install "BakerGroupInference" "C:\BakerGroup\LLM\venv\Scripts\python.exe"  
nssm.exe set "BakerGroupInference" AppParameters "C:\BakerGroup\LLM\inference\main.py"  
nssm.exe set "BakerGroupInference" AppDirectory "C:\BakerGroup\LLM\inference"  
nssm.exe set "BakerGroupInference" DisplayName "Baker Group LLM Inference Service"  
nssm.exe set "BakerGroupInference" Description "Local LLM inference processing for Baker Group Asset/Liability Management"  
  
REM Configure service user (use dedicated service account)  
nssm.exe set "BakerGroupInference" ObjectName ".\BakerGroupLLM" "SecurePassword123!"  
  
REM Set startup type and failure actions  
nssm.exe set "BakerGroupInference" Start SERVICE\_AUTO\_START  
nssm.exe set "BakerGroupInference" Type SERVICE\_WIN32\_OWN\_PROCESS  
  
REM Configure failure actions - restart service on failure  
nssm.exe set "BakerGroupInference" AppExit Default Restart  
nssm.exe set "BakerGroupInference" AppRestartDelay 5000  
nssm.exe set "BakerGroupInference" AppStdout "C:\BakerGroup\LLM\logs\inference-stdout.log"  
nssm.exe set "BakerGroupInference" AppStderr "C:\BakerGroup\LLM\logs\inference-stderr.log"  
  
REM Set environment variables  
nssm.exe set "BakerGroupInference" AppEnvironmentExtra ^  
 "PYTHONPATH=C:\BakerGroup\LLM\inference" ^  
 "OLLAMA\_HOST=127.0.0.1" ^  
 "OLLAMA\_PORT=11434" ^  
 "DATABASE\_PATH=C:\BakerGroup\LLM\data\queue.db" ^  
 "LOG\_LEVEL=INFO" ^  
 "CUDA\_VISIBLE\_DEVICES=0"  
  
REM Configure process priority and affinity  
nssm.exe set "BakerGroupInference" AppPriority NORMAL\_PRIORITY\_CLASS  
nssm.exe set "BakerGroupInference" AppAffinity All  
  
echo Service installed successfully. Starting service...  
net start "BakerGroupInference"  
  
pause

**Inference Service Entry Point**

# C:\BakerGroup\LLM\inference\main.py  
import os  
import sys  
import asyncio  
import logging  
import signal  
from pathlib import Path  
  
# Configure logging for Windows Service  
logging.basicConfig(  
 level=logging.INFO,  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  
 handlers=[  
 logging.FileHandler('C:/BakerGroup/LLM/logs/inference-service.log'),  
 logging.StreamHandler(sys.stdout)  
 ]  
)  
  
logger = logging.getLogger(\_\_name\_\_)  
  
# Add project root to path  
project\_root = Path(\_\_file\_\_).parent  
sys.path.insert(0, str(project\_root))  
  
from inference\_service import InferenceService  
  
class WindowsServiceRunner:  
 def \_\_init\_\_(self):  
 self.inference\_service = InferenceService()  
 self.running = False  
   
 async def start(self):  
 """Start the inference service"""  
 logger.info("Starting Baker Group Inference Service...")  
   
 try:  
 # Initialize Ollama and load models  
 await self.inference\_service.initialize()  
   
 self.running = True  
 logger.info("Inference service started successfully")  
   
 # Start the main processing loop  
 await self.inference\_service.run()  
   
 except Exception as e:  
 logger.error(f"Failed to start inference service: {e}")  
 raise  
   
 async def stop(self):  
 """Graceful shutdown"""  
 logger.info("Stopping Baker Group Inference Service...")  
 self.running = False  
 await self.inference\_service.cleanup()  
 logger.info("Inference service stopped")  
   
 def signal\_handler(self, signum, frame):  
 """Handle Windows service stop signals"""  
 logger.info(f"Received signal {signum}, initiating graceful shutdown...")  
 asyncio.create\_task(self.stop())  
  
async def main():  
 """Main entry point for Windows service"""  
 service\_runner = WindowsServiceRunner()  
   
 # Register signal handlers for graceful shutdown  
 signal.signal(signal.SIGTERM, service\_runner.signal\_handler)  
 signal.signal(signal.SIGINT, service\_runner.signal\_handler)  
   
 try:  
 await service\_runner.start()  
 except KeyboardInterrupt:  
 logger.info("Received keyboard interrupt")  
 except Exception as e:  
 logger.error(f"Service error: {e}")  
 sys.exit(1)  
 finally:  
 await service\_runner.stop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Run the service  
 try:  
 asyncio.run(main())  
 except Exception as e:  
 logger.error(f"Failed to start service: {e}")  
 sys.exit(1)

**Windows Server Security Configuration**

**Service Account Setup**

# create\_service\_account.ps1  
# Run as Administrator  
  
# Create dedicated service account  
$SecurePassword = ConvertTo-SecureString "SecurePassword123!" -AsPlainText -Force  
New-LocalUser -Name "BakerGroupLLM" -Password $SecurePassword -Description "Baker Group LLM Service Account" -AccountNeverExpires -PasswordNeverExpires  
  
# Grant necessary permissions  
$ServiceAccount = ".\BakerGroupLLM"  
  
# Grant "Log on as a service" right  
$tempPath = [System.IO.Path]::GetTempPath()  
$import = Join-Path -Path $tempPath -ChildPath "import.inf"  
$export = Join-Path -Path $tempPath -ChildPath "export.inf"  
$secedit = Join-Path -Path ([System.Environment]::GetFolderPath([System.Environment+SpecialFolder]::System)) -ChildPath "secedit.exe"  
  
secedit /export /cfg $export  
$contents = Get-Content -Path $export  
$contents = $contents -replace "SeServiceLogonRight = ", "SeServiceLogonRight = $ServiceAccount,"  
$contents | Set-Content -Path $import  
secedit /import /cfg $import /db secedit.sdb  
secedit /configure /db secedit.sdb  
  
# Set folder permissions  
icacls "C:\BakerGroup\LLM" /grant "${ServiceAccount}:(OI)(CI)F" /T  
icacls "C:\BakerGroup\LLM\logs" /grant "${ServiceAccount}:(OI)(CI)F" /T  
icacls "C:\BakerGroup\LLM\data" /grant "${ServiceAccount}:(OI)(CI)F" /T  
  
Write-Host "Service account created and configured successfully"

**Firewall Configuration**

REM configure\_firewall.bat  
REM Run as Administrator  
  
REM Allow internal communication between services  
netsh advfirewall firewall add rule name="Baker Group API Service" dir=in action=allow protocol=TCP localport=8000  
netsh advfirewall firewall add rule name="Baker Group Inference Service" dir=in action=allow protocol=TCP localport=8001  
netsh advfirewall firewall add rule name="Ollama Server" dir=in action=allow protocol=TCP localport=11434  
  
REM Allow HTTPS for web interface  
netsh advfirewall firewall add rule name="Baker Group HTTPS" dir=in action=allow protocol=TCP localport=443  
  
echo Firewall rules configured successfully

**Deployment Automation**

**Complete Deployment Script**

# deploy\_baker\_group\_llm.ps1  
# Complete deployment automation script  
  
param(  
 [Parameter(Mandatory=$true)]  
 [string]$InstallPath = "C:\BakerGroup\LLM"  
)  
  
Write-Host "Starting Baker Group LLM Deployment..." -ForegroundColor Green  
  
# Create directory structure  
New-Item -ItemType Directory -Path "$InstallPath\api" -Force  
New-Item -ItemType Directory -Path "$InstallPath\inference" -Force  
New-Item -ItemType Directory -Path "$InstallPath\data" -Force  
New-Item -ItemType Directory -Path "$InstallPath\logs" -Force  
New-Item -ItemType Directory -Path "$InstallPath\models" -Force  
New-Item -ItemType Directory -Path "$InstallPath\venv" -Force  
  
# Install Python dependencies  
Write-Host "Installing Python dependencies..." -ForegroundColor Yellow  
& "$InstallPath\venv\Scripts\pip.exe" install fastapi uvicorn[standard] pydantic sqlalchemy asyncio-sqlite httpx websockets ollama  
  
# Copy application files  
Copy-Item -Path ".\api\\*" -Destination "$InstallPath\api" -Recurse -Force  
Copy-Item -Path ".\inference\\*" -Destination "$InstallPath\inference" -Recurse -Force  
Copy-Item -Path ".\web.config" -Destination "$InstallPath\api\web.config" -Force  
  
# Create service account  
& .\create\_service\_account.ps1  
  
# Install and configure NSSM service  
Write-Host "Installing NSSM inference service..." -ForegroundColor Yellow  
& .\install\_inference\_service.bat  
  
# Configure IIS  
Write-Host "Configuring IIS..." -ForegroundColor Yellow  
  
# Enable required IIS features  
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerRole, IIS-WebServer, IIS-CommonHttpFeatures, IIS-HttpErrors, IIS-HttpLogging, IIS-RequestFiltering, IIS-StaticContent, IIS-Security, IIS-RequestFiltering, IIS-DefaultDocument, IIS-DirectoryBrowsing, IIS-ASPNET45  
  
# Install HttpPlatformHandler  
$platformHandlerUrl = "https://download.microsoft.com/download/C/F/F/CFF3A0B8-99D4-41A2-AE1A-496C08BEB904/HttpPlatformHandler\_amd64.msi"  
$platformHandlerPath = "$env:TEMP\HttpPlatformHandler\_amd64.msi"  
Invoke-WebRequest -Uri $platformHandlerUrl -OutFile $platformHandlerPath  
Start-Process msiexec.exe -ArgumentList "/i", $platformHandlerPath, "/quiet", "/norestart" -Wait  
  
# Create IIS application  
Import-Module WebAdministration  
New-WebAppPool -Name "BakerGroupLLMAPI"  
Set-ItemProperty -Path "IIS:\AppPools\BakerGroupLLMAPI" -Name "processModel.identityType" -Value "ApplicationPoolIdentity"  
Set-ItemProperty -Path "IIS:\AppPools\BakerGroupLLMAPI" -Name "processModel.idleTimeout" -Value "00:00:00"  
  
New-WebApplication -Site "Default Web Site" -Name "bakergroup-llm" -PhysicalPath "$InstallPath\api" -ApplicationPool "BakerGroupLLMAPI"  
  
# Configure SSL certificate (self-signed for internal use)  
$cert = New-SelfSignedCertificate -DnsName "localhost", "bakergroup-llm.local" -CertStoreLocation "cert:\LocalMachine\My"  
New-WebBinding -Name "Default Web Site" -IPAddress "\*" -Port 443 -Protocol "https" -SslFlags 1  
$binding = Get-WebBinding -Name "Default Web Site" -Port 443 -Protocol "https"  
$binding.AddSslCertificate($cert.GetCertHashString(), "My")  
  
# Start services  
Write-Host "Starting services..." -ForegroundColor Yellow  
Start-Service "BakerGroupInference"  
Start-WebAppPool "BakerGroupLLMAPI"  
  
# Configure firewall  
& .\configure\_firewall.bat  
  
Write-Host "Deployment completed successfully!" -ForegroundColor Green  
Write-Host "API Service: https://localhost/bakergroup-llm" -ForegroundColor Cyan  
Write-Host "Service Status: Get-Service 'BakerGroupInference'" -ForegroundColor Cyan

**Service Management & Monitoring**

**Service Management Commands**

REM Service management commands for administrators  
  
REM Check service status  
sc query "BakerGroupInference"  
Get-Service "BakerGroupInference"  
  
REM Start/Stop services  
net start "BakerGroupInference"  
net stop "BakerGroupInference"  
  
REM View service configuration  
nssm dump "BakerGroupInference"  
  
REM Update service configuration  
nssm set "BakerGroupInference" AppParameters "C:\BakerGroup\LLM\inference\main.py --config production"  
  
REM View logs  
Get-Content "C:\BakerGroup\LLM\logs\inference-service.log" -Tail 50 -Wait

**Performance Monitoring Script**

# monitor\_services.ps1  
# Continuous monitoring of Baker Group LLM services  
  
while ($true) {  
 $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"  
   
 # Check service status  
 $inferenceService = Get-Service "BakerGroupInference" -ErrorAction SilentlyContinue  
 $iisAppPool = Get-WebAppPool "BakerGroupLLMAPI" -ErrorAction SilentlyContinue  
   
 # Check GPU utilization  
 $gpuInfo = nvidia-smi --query-gpu=utilization.gpu,memory.used,memory.total --format=csv,noheader,nounits  
   
 # Check process memory usage  
 $pythonProcs = Get-Process python -ErrorAction SilentlyContinue  
 $memoryUsage = ($pythonProcs | Measure-Object WorkingSet -Sum).Sum / 1MB  
   
 Write-Host "[$timestamp] Inference Service: $($inferenceService.Status), IIS Pool: $($iisAppPool.State)"  
 Write-Host "[$timestamp] GPU: $gpuInfo, Python Memory: $([math]::Round($memoryUsage, 2))MB"  
   
 # Alert on service failures  
 if ($inferenceService.Status -ne "Running") {  
 Write-Warning "Inference service is not running! Attempting restart..."  
 Start-Service "BakerGroupInference"  
 }  
   
 Start-Sleep -Seconds 30  
}

**Maintenance Procedures**

**Automated Log Rotation**

# log\_rotation.ps1  
# Schedule as daily task  
  
$LogPath = "C:\BakerGroup\LLM\logs"  
$ArchivePath = "C:\BakerGroup\LLM\logs\archive"  
$MaxAge = 30 # days  
  
# Create archive directory  
if (!(Test-Path $ArchivePath)) {  
 New-Item -ItemType Directory -Path $ArchivePath -Force  
}  
  
# Archive logs older than 7 days  
Get-ChildItem -Path $LogPath -Filter "\*.log" | Where-Object {   
 $\_.LastWriteTime -lt (Get-Date).AddDays(-7)   
} | ForEach-Object {  
 $archiveFile = "$ArchivePath\$($\_.BaseName)\_$(Get-Date -Format 'yyyyMMdd')$($\_.Extension)"  
 Move-Item $\_.FullName $archiveFile  
 Compress-Archive -Path $archiveFile -DestinationPath "$archiveFile.zip"  
 Remove-Item $archiveFile  
}  
  
# Clean up archives older than MaxAge  
Get-ChildItem -Path $ArchivePath -Filter "\*.zip" | Where-Object {   
 $\_.LastWriteTime -lt (Get-Date).AddDays(-$MaxAge)   
} | Remove-Item -Force

This Windows Server deployment architecture provides Baker Group with enterprise-grade service management, automatic failover capabilities, and comprehensive monitoring while leveraging native Windows Server features for optimal security and reliability. The combination of IIS for web service hosting and NSSM for Python service management follows Microsoft best practices for Windows Server deployments.