NipService Installation Guide

Docker & Podman Setup for Linux and Windows

Version: 1.1.0 **Last Updated:** January 27, 2026

Table of Contents

Overview

NipService is a WebAPI for validating NIP/VAT numbers (PL via MF “White List”, EU via VIES REST API). This guide covers containerized deployment with Docker or Podman on Linux and Windows.

Key facts:

- Application listens on internal port **8080**
- Typical external mapping: **5001**
- Health endpoint: `/health`
- Version endpoint: `/api/Version`
- Swagger UI: `/swagger`
- Logs written to `/app/logs/app.log`

GitHub Container Registry Authentication

Nip Service image is hosted on GitHub Container Registry (ghcr.io), you’ll need to authenticate before pulling the image.

Prerequisites

1. **GitHub Account** with access to the repository (GitHub account is free and can be easly created at https://github.com/signup)
2. **Personal Access Token (PAT)** with `read:packages` scope

Creating a GitHub Personal Access Token

1. Go to GitHub: https://github.com/settings/tokens
2. Click **"Generate new token" → "Generate new token (classic)"**
3. Configure the token:
   - **Note:** `NipService Docker Access`
   - **Expiration:** Choose appropriate duration (e.g., 90 days, 1 year, or no expiration)
   - **Scopes:** Select `read:packages` (required to pull images)
4. Click **"Generate token"**
5. **IMPORTANT:** Copy the token immediately - you won't be able to see it again!

Get access to package

After creating account please send request to grzegorz.giewon@uptime-erp.com with username and email of your account.

Authentication Methods

Method 1: Docker Login (Recommended)

Linux:

```
# Login to GitHub Container Registry
echo "YOUR_GITHUB_TOKEN" | docker login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin

# Verify login
docker login ghcr.io
```

Windows PowerShell:

```
# Set variables
$env:GITHUB_TOKEN = "YOUR_GITHUB_TOKEN"
$env:GITHUB_USERNAME = "YOUR_GITHUB_USERNAME"

# Login
$env:GITHUB_TOKEN | docker login ghcr.io -u $env:GITHUB_USERNAME --password-stdin

# Clear token from environment for security
Remove-Item Env:\GITHUB_TOKEN
```

Success message:

```
Login Succeeded
```

Method 2: Podman Login

Linux:

```
# Login to GitHub Container Registry
echo "YOUR_GITHUB_TOKEN" | podman login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin

# Verify login
podman login ghcr.io
```

Windows PowerShell:

```
# Login
$env:GITHUB_TOKEN = "YOUR_GITHUB_TOKEN"
$env:GITHUB_TOKEN | podman login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin
Remove-Item Env:\GITHUB_TOKEN
```

Method 3: Using Credentials File (Advanced)

Linux:

```
# Docker credentials are stored in:
~/.docker/config.json

# Podman credentials are stored in:
~/.config/containers/auth.json
```

Windows:

```
# Docker credentials location:
C:\Users\YourUsername\.docker\config.json

# Podman credentials location:
C:\Users\YourUsername\.config\containers\auth.json
```

Updating docker-compose.yml for GitHub Container Registry

The Nip Service image is hosted on GitHub Container Registry at:

```
ghcr.io/ggiewon/nipservice-webapi:develop
```

Update the image reference in your `docker-compose.yml`:

```
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    # or for specific version (when available):
    # image: ghcr.io/ggiewon/nipservice-webapi:1.0.0
    container_name: nipservice-api
```

```
    ports:
      - "5001:8080"
    # ... rest of configuration
```

Complete example:

```yaml
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    container_name: nipservice-api
    ports:
      - "5001:8080"
    volumes:
      - ./Logs:/app/logs
      # Optional: mount additional config (e.g., for proxy settings)
      # - ./appsettings.additional.json:/app/appsettings.additional.json:ro
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s
    networks:
      - nipservice-network

networks:
  nipservice-network:
    driver: bridge
```

Pulling the Image After Authentication

Docker:

```
# Pull the Nip Service image
docker pull ghcr.io/ggiewon/nipservice-webapi:develop

# Or use docker-compose (will use authenticated session)
docker-compose pull
```

Podman:

```
# Pull the Nip Service image
podman pull ghcr.io/ggiewon/nipservice-webapi:develop
```

```
# Or use podman-compose
podman-compose pull
```

Troubleshooting Authentication

Error: "unauthorized: authentication required"

Solution:

```
# Verify you're logged in
docker login ghcr.io
# or
podman login ghcr.io

# If not logged in, authenticate again with your token
echo "YOUR_TOKEN" | docker login ghcr.io -u YOUR_USERNAME --password-stdin
```

Error: "denied: permission_denied"

Possible causes:

1. Token doesn't have `read:packages` permission
2. You don't have access to the repository
3. Repository is private and you're not a collaborator

Solution:

- Generate a new token with correct permissions
- Contact repository owner for access
- Verify repository visibility settings

Error: "Error response from daemon: Get https://ghcr.io/v2/: denied"

Solution (Linux):

```
# Clear old credentials
rm ~/.docker/config.json

# Login again
docker login ghcr.io
```

Solution (Windows):

```
# Clear credentials
Remove-Item "$env:USERPROFILE\.docker\config.json"

# Login again
docker login ghcr.io
```

Verify Authentication Status

```
# Docker
cat ~/.docker/config.json | grep ghcr.io

# Podman (Linux)
cat ~/.config/containers/auth.json | grep ghcr.io

# Windows PowerShell (Docker)
Get-Content "$env:USERPROFILE\.docker\config.json" | Select-String "ghcr.io"
```

Security Best Practices

1. **Never commit tokens to Git**

```
# Add to .gitignore
echo "*.token" >> .gitignore
echo ".env" >> .gitignore
```

2. **Use environment variables**

```
# Linux - add to ~/.bashrc or ~/.profile
export GITHUB_TOKEN="your_token_here"

# Windows - use User Environment Variables
# Settings → System → About → Advanced system settings → Environment Variables
```

3. **Rotate tokens regularly**
   - Set expiration dates on tokens
   - Generate new tokens every 90 days
   - Revoke old tokens after rotation

4. **Use minimal permissions**
   - Only grant `read:packages` for pulling images
   - Don't use tokens with `write:packages` or `delete:packages` unless necessary

5. **Store tokens securely**

```
# Linux - use secret management tools
# Store in password manager or system keyring

# Example using pass (password manager)
pass insert github/docker-token
pass show github/docker-token | docker login ghcr.io -u username --password-stdin
```

Logout

When finished or to clear credentials:

```
# Docker
docker logout ghcr.io

# Podman
podman logout ghcr.io
```

Alternative: Using Image as .tar File

If you cannot authenticate or prefer offline distribution:

1. **On machine with access:**

   ```
   # Pull and save image
   docker pull ghcr.io/ggiewon/nipservice-webapi:develop
   docker save ghcr.io/ggiewon/nipservice-webapi:develop -o nipservice-webapi-develop.tar

   # Compress (optional)
   gzip nipservice-webapi-develop.tar
   ```

2. **Transfer file to target machine** (USB, SCP, etc.)

3. **On target machine:**

   ```
   # Load image
   docker load -i nipservice-webapi-develop.tar
   # or if compressed:
   docker load -i nipservice-webapi-develop.tar.gz

   # Tag image if needed (only if using local name in docker-compose.yml)
   docker tag ghcr.io/ggiewon/nipservice-webapi:develop nipservice-webapi:develop
   ```

Linux Installation

Prerequisites

Update your system first:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Linux Docker

Step 1: Install Docker and Docker Compose

```
# Install Docker
sudo apt-get install -y docker.io docker-compose

# Start Docker service
sudo systemctl start docker
sudo systemctl enable docker

# Add current user to docker group (to avoid using sudo)
sudo usermod -aG docker $USER

# IMPORTANT: Log out and log back in for group changes to take effect
```

Step 2: Verify Installation

After logging back in:

```
docker --version
docker-compose --version
```

Expected output:

```
Docker version 24.x.x, build xxxxxxx
docker-compose version 1.29.x, build xxxxxxx
```

Step 3: Create Working Directory

```
# Create application directory
mkdir -p ~/NipService
cd ~/NipService
```

Step 4: Create docker-compose.yml File

```
nano docker-compose.yml
```

Paste the following content:

```
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    container_name: nipservice-api
    ports:
      - "5001:8080"
    volumes:
      - ./Logs:/app/logs
      # Optional: mount additional config (e.g., for proxy settings)
```

```
      # - ./appsettings.additional.json:/app/appsettings.additional.json:ro
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s
    networks:
      - nipservice-network

networks:
  nipservice-network:
    driver: bridge
```

Save and exit (Ctrl+X, then Y, then Enter).

Note: If you need proxy settings, uncomment the `appsettings.additional.json` volume line and create the file as described in Step 5.1.

Step 5: Create Log Directory

```
# Create Logs directory
mkdir -p Logs
```

Set proper permissions:

```
chmod 755 Logs/
```

Step 5.1 (optional) Configure proxy settings

If your environment requires HTTP proxy for outgoing connections, create additional configuration file:

```
nano appsettings.additional.json
```

Paste the following content (adjust values as needed):

```
{
  "Validation": {
    "Proxy": {
      "Enabled": true,
      "Address": "http://proxy.example.com",
      "Port": 8080,
      "Username": null,
      "Password": null,
      "BypassOnLocal": true,
```

```
      "BypassList": []
    }
  }
}
```

Configuration options:

Option	Type	Default	Description
`Enabled`	bool	`false`	Enable/disable proxy
`Address`	string	`""`	Proxy server URL (e.g., `http://proxy.company.com`)
`Port`	int	`8080`	Proxy server port
`Username`	string	`null`	Username for proxy authentication (optional)
`Password`	string	`null`	Password for proxy authentication (optional)
`BypassOnLocal`	bool	`true`	Bypass proxy for local addresses
`BypassList`	string[]	`[]`	List of addresses to bypass (e.g., `["*.local", "10.*"]`)

Set proper permissions:

```
chmod 644 appsettings.additional.json
```

Note: The proxy settings apply to all external API calls (Biała Lista MF, VIES).

Step 6: Verify Directory Structure

```
tree -L 2
# Or without tree:
ls -la
```

Expected structure:

```
NipService/
├── docker-compose.yml
└── appsettings.additional.json (optional)
```

Step 7: Load Docker Image

Important: The Nip Service image is hosted on GitHub Container Registry and requires authentication. See the [GitHub Container Registry Authentication] section above.

Option A: Pull from GitHub Container Registry (RECOMMENDED - requires authentication):

```
# First, authenticate (if not already done)
echo "YOUR_GITHUB_TOKEN" | docker login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin
```

```
# Pull the image using docker-compose
docker-compose pull

# Or pull directly
docker pull ghcr.io/ggiewon/nipservice-webapi:develop
```

Option B: Load from .tar file (offline method):

```
# If you received the image as a tar file
docker load -i nipservice-webapi-develop.tar

# Verify image loaded
docker images | grep nipservice
```

Step 8: Start the Application

```
docker-compose up -d
```

Parameters explanation:

- `up` - starts services defined in docker-compose.yml
- `-d` - runs in detached mode (background)

Step 9: Verify Application is Running

```
# Check container status
docker-compose ps

# Expected output:
# NAME              IMAGE                          STATUS          PORTS
# nipservice-api     nipservice-webapi:develop     Up X seconds    0.0.0.0:5001->8080/tcp
```

Step 10: Test the Application

```
# Health check
curl http://localhost:5001/health

# Application version
curl http://localhost:5001/api/Version
```

In web browser, navigate to:

- Swagger UI: http://localhost:5001/swagger
- Health Check: http://localhost:5001/health

Linux Podman

Step 1: Install Podman

```
# Ubuntu 20.10 and newer
sudo apt-get install -y podman

# For older Ubuntu versions, add repository first:
sudo sh -c "echo 'deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable
wget -nv https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/xUbuntu_$(lsb_r
sudo apt-get update
sudo apt-get install -y podman

# Install podman-compose
sudo apt-get install -y python3-pip
pip3 install podman-compose
```

Step 2: Verify Installation

```
podman --version
podman-compose --version
```

Step 3: Create Working Directory

```
mkdir -p ~/NipService
cd ~/NipService
```

Step 4: Create podman-compose.yml File

```
nano podman-compose.yml
```

Paste the following content:

```
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    container_name: nipservice-api
    ports:
      - "5001:8080"
    volumes:
      - ./Logs:/app/logs
      # Optional: mount additional config (e.g., for proxy settings)
      # - ./appsettings.additional.json:/app/appsettings.additional.json:ro
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
```

```
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s

networks:
  nipservice-network:
    driver: bridge
```

Note: If you need proxy settings, uncomment the `appsettings.additional.json` volume line and create the file as described in Step 5.1.

Step 5: Create Logs Directory

```
mkdir -p Logs
```

Step 5.1 (optional) Configure proxy settings

If your environment requires HTTP proxy, follow the same steps as described in Linux Docker Step 5.1.

Step 6: Load Image

Important: The Nip Service image is hosted on GitHub Container Registry and requires authentication. See the GitHub Container Registry Authentication section.

Option A: Pull from GitHub Container Registry (RECOMMENDED - requires authentication):

```
# Authenticate
echo "YOUR_GITHUB_TOKEN" | podman login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin

# Pull the image using podman-compose
podman-compose pull

# Or pull directly
podman pull ghcr.io/ggiewon/nipservice-webapi:develop
```

Option B: Load from .tar file (offline method):

```
podman load -i nipservice-webapi-develop.tar
podman images | grep nipservice
```

Step 7: Start the Application

```
podman-compose up -d
```

Step 8: Verify and Test

```
# Check status
podman-compose ps

# View logs
podman-compose logs -f nipservice-api

# Test endpoints
curl http://localhost:5001/health
curl http://localhost:5001/api/Version
```

Step 9: Configure Systemd Service (Optional - Auto-start on Boot)

```
# Generate systemd service file
cd ~/NipService
podman generate systemd --name nipservice-api --files --new

# Move to systemd directory
mkdir -p ~/.config/systemd/user/
mv container-nipservice-api.service ~/.config/systemd/user/

# Enable and start service
systemctl --user enable container-nipservice-api.service
systemctl --user start container-nipservice-api.service

# Enable lingering (allows services to run without login)
loginctl enable-linger $USER
```

Windows Installation

Windows Docker

> ⚠️ **IMPORTANT LICENSING NOTICE**
>
> **Docker Desktop is FREE for:**
>
> - Personal use
> - Small businesses (fewer than 250 employees AND less than $10 million in annual revenue)
> - Education and non-commercial open source projects
>
> **Docker Desktop requires a PAID SUBSCRIPTION for:**
>
> - Large businesses (250+ employees OR $10 million+ annual revenue)
> - Commercial use in medium/large organizations
>
> **Pricing:** Starting at $5/user/month (Professional) or $7/user/month (Team)
>
> For detailed licensing information, visit: https://www.docker.com/pricing/
>
> **Alternative:** If your organization requires a free solution, consider using **Podman Desktop** (see Windows Podman section below) which is completely free and open source.

Step 1: Install Docker Desktop

1. Download Docker Desktop from: https://www.docker.com/products/docker-desktop
2. Run the installer: `Docker Desktop Installer.exe`
3. Follow the installation wizard
4. Restart your computer when prompted
5. Launch Docker Desktop from Start menu
6. Wait for Docker to start (the whale icon in system tray will stop animating)

Step 2: Verify Installation

Open PowerShell and run:

```
docker --version
docker-compose --version
```

Expected output:

```
Docker version 24.x.x, build xxxxxxx
Docker Compose version v2.x.x
```

Step 3: Create Working Directory

```
# Create directory
New-Item -Path "C:\NipService" -ItemType Directory
Set-Location -Path "C:\NipService"
```

Step 4: Create docker-compose.yml File

```
# Create file using notepad
notepad docker-compose.yml
```

Paste the following content:

```
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    container_name: nipservice-api
    ports:
      - "5001:8080"
    volumes:
      - ./Logs:/app/logs
      # Optional: mount additional config (e.g., for proxy settings)
      # - ./appsettings.additional.json:/app/appsettings.additional.json:ro
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
```

```
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s
    networks:
      - nipservice-network

networks:
  nipservice-network:
    driver: bridge
```

Save and close Notepad.

Note: If you need proxy settings, uncomment the `appsettings.additional.json` volume line and create the file as described in Step 5.1.

Step 5: Create Logs Directory

```
# Create Logs directory
New-Item -Path ".\Logs" -ItemType Directory

# Verify structure
Get-ChildItem -Recurse
```

Expected structure:

```
C:\NipService\
├── docker-compose.yml
```

Step 5.1 (optional) Configure proxy settings

If your environment requires HTTP proxy for outgoing connections:

```
notepad appsettings.additional.json
```

Paste the following content (adjust values as needed):

```
{
  "Validation": {
    "Proxy": {
      "Enabled": true,
      "Address": "http://proxy.example.com",
      "Port": 8080,
      "Username": null,
      "Password": null,
      "BypassOnLocal": true,
      "BypassList": []
    }
```

```
  }
}
```

Save and close Notepad. Then uncomment the volume line in `docker-compose.yml`:

```
volumes:
  - ./Logs:/app/logs
  - ./appsettings.additional.json:/app/appsettings.additional.json:ro
```

Step 6: Load Docker Image

Important: The Nip Service image is hosted on GitHub Container Registry and requires authentication. See the GitHub Container Registry Authentication section.

Option A: Pull from GitHub Container Registry (RECOMMENDED - requires authentication):

```
# Authenticate
$env:GITHUB_TOKEN = "YOUR_GITHUB_TOKEN"
$env:GITHUB_TOKEN | docker login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin
Remove-Item Env:\GITHUB_TOKEN

# Pull the image using docker-compose
docker-compose pull

# Or pull directly
docker pull ghcr.io/ggiewon/nipservice-webapi:develop
```

Option B: Load from .tar file (offline method):

```
docker load -i nipservice-webapi-develop.tar
docker images | Select-String nipservice
```

Step 7: Start the Application

```
docker-compose up -d
```

Step 8: Verify Application is Running

```
# Check container status
docker-compose ps

# Check logs
docker-compose logs nipservice-api
```

Step 9: Test the Application

```powershell
# Health check
Invoke-WebRequest -Uri http://localhost:5001/health

# Application version
Invoke-WebRequest -Uri http://localhost:5001/api/Version

# Or use curl (if installed)
curl http://localhost:5001/health
```

In web browser:

- Swagger UI: http://localhost:5001/swagger
- Health Check: http://localhost:5001/health

Windows Podman

> ✅ **FREE AND OPEN SOURCE**
>
> Podman Desktop is completely **free** for all users and use cases:
>
> - No licensing restrictions
> - No commercial subscriptions required
> - Open source (Apache 2.0 license)
> - Suitable for personal, educational, and commercial use
>
> **Perfect alternative to Docker Desktop for organizations requiring free solutions.**

Step 1: Install Podman Desktop

1. Download Podman Desktop from: https://podman-desktop.io/downloads
2. Run the installer: `podman-desktop-x.x.x-setup.exe`
3. Follow the installation wizard
4. Launch Podman Desktop from Start menu
5. Complete the initial setup wizard
   - Initialize Podman Machine
   - Set resource limits (CPU, Memory)

Step 2: Install Podman CLI

Podman Desktop includes the CLI, but verify it's in your PATH:

```powershell
# Check Podman version
podman --version

# If command not found, add to PATH:
# Typically located at: C:\Program Files\RedHat\Podman\
$env:Path += ";C:\Program Files\RedHat\Podman\"
```

Step 3: Start Podman Machine

```
# Initialize Podman machine (first time only)
podman machine init

# Start Podman machine
podman machine start

# Verify it's running
podman machine list
```

Expected output:

```
NAME                     VM TYPE     CREATED      LAST UP            CPUS        MEMORY      DISK SIZE
podman-machine-default*  wsl         2 hours ago  Currently running  2           2GiB        100GiB
```

Step 4: Create Working Directory

```
New-Item -Path "C:\NipService" -ItemType Directory
Set-Location -Path "C:\NipService"
```

Step 5: Create Compose File

```
notepad docker-compose.yml
```

Note: Podman uses the same docker-compose.yml format. Paste:

```
services:
  nipservice-api:
    image: ghcr.io/ggiewon/nipservice-webapi:develop
    container_name: nipservice-api
    ports:
      - "5001:8080"
    volumes:
      - ./Logs:/app/logs
      # Optional: mount additional config (e.g., for proxy settings)
      # - ./appsettings.additional.json:/app/appsettings.additional.json:ro
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s
```

Note: If you need proxy settings, uncomment the `appsettings.additional.json` volume line and create the file as described in Step 5.1.

Step 6: Create Logs Directory

```
New-Item -Path ".\Logs" -ItemType Directory
```

Step 6.1 (optional) Configure proxy settings

If your environment requires HTTP proxy:

```
notepad appsettings.additional.json
```

Paste the following content:

```
{
  "Validation": {
    "Proxy": {
      "Enabled": true,
      "Address": "http://proxy.example.com",
      "Port": 8080,
      "Username": null,
      "Password": null,
      "BypassOnLocal": true,
      "BypassList": []
    }
  }
}
```

Save and close Notepad. Then uncomment the volume line in `docker-compose.yml`.

Step 7: Load Image

Important: The Nip Service image is hosted on GitHub Container Registry and requires authentication. See the [GitHub Container Registry Authentication] section.

Option A: Pull from GitHub Container Registry (RECOMMENDED - requires authentication):

```
# Authenticate
$env:GITHUB_TOKEN = "YOUR_GITHUB_TOKEN"
$env:GITHUB_TOKEN | podman login ghcr.io -u YOUR_GITHUB_USERNAME --password-stdin
Remove-Item Env:\GITHUB_TOKEN

# Pull the image using podman-compose
podman-compose pull

# Or pull directly
podman pull ghcr.io/ggiewon/nipservice-webapi:develop
```

Option B: Load from .tar file (offline method):

```
podman load -i nipservice-webapi-develop.tar
podman images | Select-String nipservice
```

Step 8: Start the Application

```
# Using podman-compose
podman-compose up -d

# Or using podman directly
podman play kube docker-compose.yml
```

Step 9: Verify and Test

```
# Check status
podman ps

# View logs
podman logs nipservice-api

# Follow logs
podman logs -f nipservice-api

# Test endpoints
Invoke-WebRequest -Uri http://localhost:5001/health
```

Common Operations

Viewing Logs

Docker:

```
# Linux
docker-compose logs -f nipservice-api
docker-compose logs --tail=100 nipservice-api
docker-compose logs --since 1h nipservice-api

# Windows PowerShell
docker-compose logs -f nipservice-api
docker-compose logs --tail 100 nipservice-api
```

Podman:

```
# Linux
podman-compose logs -f nipservice-api
podman logs -f nipservice-api

# Windows PowerShell
podman logs -f nipservice-api
podman logs --tail 100 nipservice-api
```

Stopping the Application

Docker:

```
# Stop without removing container
docker-compose stop

# Stop and remove container (logs in Logs/ remains)
docker-compose down

# Stop and remove everything including volumes
docker-compose down -v
```

Podman:

```
# Stop without removing
podman-compose stop

# Stop and remove
podman-compose down

# Using podman directly
podman stop nipservice-api
podman rm nipservice-api
```

Restarting the Application

Docker:

```
# Restart running container
docker-compose restart

# Start after stop
docker-compose start

# Recreate container
docker-compose up -d --force-recreate
```

Podman:

```
podman-compose restart
podman-compose start
podman restart nipservice-api
```

Updating the Application

Docker:

```
# 1. Stop and remove old container
docker-compose down

# 2. Pull new image
docker-compose pull

# 3. Start new version
docker-compose up -d

# 4. Check logs
docker-compose logs -f nipservice-api
```

Podman:

```
# 1. Stop container
podman-compose down

# 2. Pull new image
podman-compose pull

# 3. Start new version
podman-compose up -d

# 4. Verify
podman-compose ps
```

Checking Resource Usage

Docker:

```
# Real-time stats
docker stats nipservice-api

# Container details
docker inspect nipservice-api

# Processes inside container
docker top nipservice-api
```

Podman:

```
# Real-time stats
podman stats nipservice-api

# Container details
podman inspect nipservice-api

# Processes
podman top nipservice-api
```

Configuration Management

External Configuration (appsettings.additional.json)

NipService supports external configuration through `appsettings.additional.json` file. This file is loaded at startup and can override any settings from the default configuration. It's particularly useful for:

- Configuring HTTP proxy settings
- Adjusting timeout values
- Changing cache TTL settings

Creating the configuration file:

```
# Linux
nano appsettings.additional.json

# Windows
notepad appsettings.additional.json
```

Example - Proxy configuration:

```
{
  "Validation": {
    "Proxy": {
      "Enabled": true,
      "Address": "http://proxy.company.com",
      "Port": 8080,
      "Username": "proxyuser",
      "Password": "proxypass",
      "BypassOnLocal": true,
      "BypassList": ["*.local", "10.*", "192.168.*"]
    }
  }
}
```

Example - Custom timeout and cache settings:

```
{
  "Validation": {
    "Whitelist": {
      "TimeoutSeconds": 30,
      "RetryCount": 5
    },
    "Cache": {
      "PolishTtlSeconds": 7200,
      "ViesTtlSeconds": 7200
    }
  }
}
```

Mounting in Docker/Podman:

```
volumes:
  - ./appsettings.additional.json:/app/appsettings.additional.json:ro
```

Note: Changes to this file require container restart to take effect.

Update via API

```
# Linux
curl -X POST http://localhost:5001/api/Authorization/upload \
  -H "Content-Type: application/json" \
  -d '{
    "nips": ["5732296089", "9721121810"],
    "version": "1.0",
    "createdAt": "2024-12-29T12:00:00Z",
    "description": "Production NIP list",
    "hash": "your-hmac-sha256-hash-in-base64"
  }'
```

Windows PowerShell:

```
$body = @{
    nips = @("5732296089", "9721121810")
    version = "1.0"
    createdAt = "2024-12-29T12:00:00Z"
    description = "Production NIP list"
    hash = "your-hmac-sha256-hash-in-base64"
} | ConvertTo-Json

Invoke-RestMethod -Uri "http://localhost:5001/api/Authorization/upload" `
    -Method Post `
    -Body $body `
    -ContentType "application/json"
```

Retrieve Current NIP List

```
# JSON format
curl http://localhost:5001/api/Authorization

# Text format (one NIP per line)
curl http://localhost:5001/api/Authorization/NipListFile

# Save to file
curl http://localhost:5001/api/Authorization/NipListFile > nip-list.txt
```

Windows PowerShell:

```
# JSON format
Invoke-RestMethod -Uri "http://localhost:5001/api/Authorization"

# Text format
Invoke-WebRequest -Uri "http://localhost:5001/api/Authorization/NipListFile" `
    -OutFile "nip-list.txt"
```

Troubleshooting

Problem: Port 5001 Already in Use

Linux - Check what's using the port:

```
sudo netstat -tlnp | grep 5001
# or
sudo lsof -i :5001
```

Solution 1: Stop the application using port 5001

Solution 2: Change port in docker-compose.yml:

```
ports:
  - "5001:8080"  # Changed from 5001 to 5001
```

Then restart:

```
docker-compose down
docker-compose up -d
```

Windows - Check port usage:

```
Get-NetTCPConnection -LocalPort 5001
# or
netstat -ano | findstr :5001
```

Problem: Container Won't Start

Check logs:

```
# Docker
docker-compose logs nipservice-api

# Podman
podman-compose logs nipservice-api
```

Check container details:

```
# Docker
docker inspect nipservice-api

# Podman
podman inspect nipservice-api
```

Verify Docker/Podman is running:

```
# Linux Docker
sudo systemctl status docker

# Restart if needed
sudo systemctl restart docker

# Windows - restart Docker Desktop or Podman Desktop from system tray
```

Problem: Permission Denied (Linux)

For Docker:

```
# Check if user is in docker group
groups

# If "docker" is not in the list:
sudo usermod -aG docker $USER

# Log out and log back in, or use:
newgrp docker
```

```
# Temporarily use sudo (not recommended for regular use)
sudo docker-compose up -d
```

For Podman:

```
# Podman runs rootless by default, no special permissions needed
# If issues persist, try:
podman system migrate
```

Problem: Application Not Responding

Check if container is running:

```
# Docker
docker-compose ps
docker ps -a | grep nipservice

# Podman
podman-compose ps
podman ps -a | grep nipservice
```

Check if port is open:

```
# Linux
netstat -tlnp | grep 5001

# Windows
Test-NetConnection localhost -Port 5001
```

Test from inside container:

```
# Docker
docker exec nipservice-api curl http://localhost:8080/health

# Podman
podman exec nipservice-api curl http://localhost:8080/health
```

Check health status:

```
# Docker
docker inspect --format='{{.State.Health.Status}}' nipservice-api

# Podman
podman inspect --format='{{.State.Health.Status}}' nipservice-api
```

Problem: Container Keeps Restarting

View recent logs:

```
docker-compose logs --tail=100 nipservice-api
```

Disable auto-restart temporarily and run interactively:

```
docker-compose down
docker-compose up
# (without -d flag to see logs in real-time)
```

Common causes:

- Missing or invalid configuration file
- Port conflicts
- Insufficient resources (CPU/RAM)
- Corrupted image

Problem: Cannot Access Swagger UI

Verify application is running:

```
curl http://localhost:5001/health
```

Check firewall (Linux):

```
sudo ufw status
sudo ufw allow 5001/tcp
```

Check Windows Defender Firewall:

```
# Create firewall rule
New-NetFirewallRule -DisplayName "Nip Service" -Direction Inbound -LocalPort 5001 -Protocol TCP -Act
```

Try accessing from different browser or clear cache

Problem: Podman Machine Not Starting (Windows)

```
# Check machine status
podman machine list

# Stop and remove existing machine
podman machine stop
podman machine rm podman-machine-default
```

```
# Reinitialize
podman machine init
podman machine start

# Check logs
podman machine inspect podman-machine-default
```

Production Considerations

Security Best Practices

Linux

```
# 1. Configure firewall
sudo ufw allow from 192.168.1.0/24 to any port 5001
sudo ufw enable

# 2. Set proper file permissions
chmod 755 Logs/

# 3. Regular updates
sudo apt-get update
sudo apt-get upgrade docker.io
```

Windows

```
# 1. Configure Windows Firewall
New-NetFirewallRule -DisplayName "Nip Service - Allow Local Network" `
    -Direction Inbound `
    -LocalPort 5001 `
    -Protocol TCP `
    -Action Allow `
    -RemoteAddress 192.168.1.0/24

# 2. Use Docker Desktop's security scanning
docker scan nipservice-webapi:develop
```

Using Reverse Proxy (Linux - Nginx)

```
# Install Nginx
sudo apt-get install nginx

# Create configuration
sudo nano /etc/nginx/sites-available/nipservice
```

Add the following configuration:

```
server {
    listen 80;
    server_name your-domain.com;

    location / {
        proxy_pass http://localhost:5001;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Enable and restart:

```
sudo ln -s /etc/nginx/sites-available/nipservice /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

SSL/TLS Configuration (Optional)

```
# Install Certbot
sudo apt-get install certbot python3-certbot-nginx

# Obtain certificate
sudo certbot --nginx -d your-domain.com

# Auto-renewal is configured automatically
```

Monitoring

Set up Log Rotation (Linux - Docker)

```
# Create Docker daemon configuration
sudo nano /etc/docker/daemon.json
```

Add:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
```

```
  }
}
```

Restart Docker:

```
sudo systemctl restart docker
```

Continuous Health Monitoring

Linux:

```
# Create monitoring script
nano ~/monitor-nipservice.sh
```

Add:

```
#!/bin/bash
while true; do
    if ! curl -f -s http://localhost:5001/health > /dev/null; then
        echo "$(date): Nip Service is down, restarting..."
        cd ~/NipService
        docker-compose restart
    fi
    sleep 60
done
```

Make executable and run:

```
chmod +x ~/monitor-nipservice.sh
nohup ~/monitor-nipservice.sh &
```

Windows (PowerShell script):

```
# Create monitoring script
while ($true) {
    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5001/health" -TimeoutSec 5
        if ($response.StatusCode -ne 200) {
            Write-Host "$(Get-Date): Nip Service unhealthy, restarting..."
            Set-Location C:\NipService
            docker-compose restart
        }
    } catch {
        Write-Host "$(Get-Date): Nip Service is down, restarting..."
        Set-Location C:\NipService
        docker-compose restart
```

```
    }
    Start-Sleep -Seconds 60
}
```

Auto-start on System Boot

Linux - Docker

Docker is already set to auto-start. The container will automatically restart due to `restart: unless-stopped` policy.

Verify:

```
sudo systemctl status docker
docker inspect nipservice-api | grep -A 5 RestartPolicy
```

Linux - Podman with Systemd

```
# Generate systemd service
cd ~/NipService
podman generate systemd --name nipservice-api --files --new

# Install service
mkdir -p ~/.config/systemd/user/
mv container-nipservice-api.service ~/.config/systemd/user/

# Enable and start
systemctl --user enable container-nipservice-api.service
systemctl --user start container-nipservice-api.service

# Enable lingering (allows service to run without login)
loginctl enable-linger $USER
```

Windows - Docker Desktop

Docker Desktop starts automatically with Windows. Containers with `restart: unless-stopped` will auto-start.

Verify in Docker Desktop:

1. Open Docker Desktop
2. Go to Settings → General
3. Ensure “Start Docker Desktop when you log in” is checked

Windows - Podman

Podman Machine can be configured to auto-start:

```
# Configure Podman machine to start on boot
podman machine set --rootful=false --now podman-machine-default
```

Resource Limits

Add resource limits to docker-compose.yml:

```yaml
services:
  nipservice-api:
    # ... existing configuration ...
    deploy:
      resources:
        limits:
          cpus: '2'
          memory: 2G
        reservations:
          cpus: '1'
          memory: 1G
```

Cleanup and Maintenance

Docker:

```bash
# Remove unused images
docker image prune -a

# Remove unused volumes
docker volume prune

# Remove all unused resources
docker system prune -a

# Check disk usage
docker system df
```

Podman:

```bash
# Remove unused images
podman image prune -a

# Remove unused volumes
podman volume prune

# System cleanup
podman system prune -a

# Check disk usage
podman system df
```

Quick Reference

Docker Commands Cheatsheet

```
# Start
docker-compose up -d

# Stop
docker-compose stop

# Remove
docker-compose down

# Restart
docker-compose restart

# Logs
docker-compose logs -f

# Status
docker-compose ps

# Execute command in container
docker exec -it nipservice-api bash

# View resource usage
docker stats nipservice-api
```

Podman Commands Cheatsheet

```
# Start
podman-compose up -d

# Stop
podman-compose stop

# Remove
podman-compose down

# Restart
podman-compose restart

# Logs
podman logs -f nipservice-api

# Status
podman ps

# Execute command in container
podman exec -it nipservice-api bash

# View resource usage
podman stats nipservice-api
```

Testing Endpoints

```
# Health check
curl http://localhost:5001/health

# Version
curl http://localhost:5001/api/Version

# Swagger UI (browser)
http://localhost:5001/swagger
```

Support and Documentation

- **Application Manual:** NipService_Manual.pdf
- **Swagger Documentation:** http://localhost:5001/swagger
- **Health Check:** http://localhost:5001/health
- **Docker Documentation:** https://docs.docker.com
- **Podman Documentation:** https://docs.podman.io

Document Version: 1.0
Last Updated: December 29, 2024
Compatible with Nip Service: 1.0.0