Python Code Executor Service

A secure, sandboxed Python code execution service built with Flask and NsJail, deployed on Google Cloud Run.

Overview

This service enables users to execute arbitrary Python code in a secure, isolated environment. It accepts Python scripts via a REST API, executes them within an NsJail sandbox, and returns the result of the `main()` function along with stdout output.

Live Service

Cloud Run URL: https://python-executor-wddqxxteba-uc.a.run.app

Features

- **Filesystem Isolation**: Read-only bind mounts prevent access to secrets and config files
- **Secure Execution**: NsJail sandboxing with isolated execution environment
- **Resource Limits**: 30-second execution timeout, 1GB memory limit
- **Crash Containment**: Isolated subprocess execution prevents service disruption
- **Syscall Filtering**: NsJail's built-in seccomp-bpf protection blocks dangerous system calls
- **Library Support**: Includes pandas, numpy, and Python standard library
- **Input Validation**: Ensures scripts contain valid `main()` function
- **JSON Response**: Structured output with results and stdout
- **Production Ready**: Deployed on Google Cloud Run with auto-scaling

Security Model

Multi-Layer Isolation

1. Filesystem Isolation (NsJail)

- Read-only bind mounts (`-R` flags) for Python libraries and system files
- Customer code cannot access `/app/` directory (secrets, config protected)
- Only `/tmp` is writable via tmpfs mount
- No access to environment variables or internal files

2. Process Isolation (NsJail)

- Execve mode (`-Me`) creates isolated subprocess for each execution
- 30-second timeout kills infinite loops and runaway processes
- Crashes contained within subprocess

3. Syscall Filtering (NsJail)

- Built-in seccomp-bpf policies block dangerous system calls
- Prevents sandbox escape attempts
- No network socket creation allowed

4. Platform Isolation (Cloud Run)

- Hardware-backed microVM isolation per container
- 1GB memory and 1 vCPU limits prevent resource exhaustion
- gVisor runtime provides kernel-level isolation

Cloud Run Constraints

Per-script resource limits cannot be enforced due to Cloud Run's `RLIMIT_RTPRIO` restriction. Resource limits are applied at the container level (1GB RAM, 1 vCPU) rather than per-script. The 30-second timeout prevents monopolization by individual scripts.

API Endpoints

GET /

Returns API information and available endpoints.

Response:

```
{
  "service": "Python Code Executor",
  "version": "1.0.0",
  "endpoints": {
    "/execute": "POST - Execute Python script",
    "/health": "GET - Health check"
  }
}
```

POST /execute

Execute a Python script and return the result.

Request:

```
{
  "script": "def main():\n    return {'message': 'Hello, World!'}"
}
```

Success Response (200):

```
{
  "result": {"message": "Hello, World!"},
  "stdout": ""
}
```

Error Response (400):

```
{
  "error": "Script must contain a 'def main()' function"
}
```

GET /health

Health check endpoint for monitoring.

Response:

```
{
  "status": "healthy",
  "service": "python-executor",
  "version": "1.0.0"
}
```

Requirements

- Script must contain a `def main()` function
- The `main()` function must return a JSON-serializable object
- Maximum execution time: 30 seconds
- Maximum script size: 100KB

Example Usage

Example 1: Simple Calculation

```
curl -X POST https://python-executor-wddqxxteba-uc.a.run.app/execute \
  -H "Content-Type: application/json" \
  -d '{
    "script": "def main():\n    result = 2 + 2\n    print(\"Calculating...\")\n    return {\"answer\": result}"
  }'
```

Response:

```
{
  "result": {"answer": 4},
  "stdout": "Calculating..."
}
```

Example 2: Using Pandas

```
curl -X POST https://python-executor-wddqxxteba-uc.a.run.app/execute \
  -H "Content-Type: application/json" \
  -d '{
    "script": "import pandas as pd\n\ndef main():\n    df = pd.DataFrame({\"a\":, \"b\": })\n    return {\"sum_a\": int(df[\"a\"].su
  }'
```

Response:

```
{
  "result": {"sum_a": 6, "sum_b": 15},
  "stdout": ""
}
```

Example 3: Using NumPy

```
curl -X POST https://python-executor-wddqxxteba-uc.a.run.app/execute \
  -H "Content-Type: application/json" \
  -d '{
    "script": "import numpy as np\n\ndef main():\n    arr = np.array()\n    return {\"mean\": float(arr.mean()), \"std\": float(arr.
  }'
```

Response:

```
{
  "result": {"mean": 3.0, "std": 1.4142135623730951},
  "stdout": ""
}
```

Example 4: Error Handling

```
curl -X POST https://python-executor-wddqxxteba-uc.a.run.app/execute \
  -H "Content-Type: application/json" \
  -d '{
    "script": "def calculate():\n    return {\"value\": 42}"
  }'
```

Response:

```
{
  "error": "Script must contain a 'def main()' function"
}
```

Local Development

Prerequisites

- Docker installed
- Google Cloud SDK (for deployment)

Run Locally

```
# Build the Docker image
docker build -t python-executor .

# Run the container
docker run -p 8080:8080 python-executor

# Test locally
curl -X POST http://localhost:8080/execute \
  -H "Content-Type: application/json" \
  -d '{"script": "def main():\n    return {\"status\": \"success\"}"}'
```

Deployment to Google Cloud Run

Prerequisites

1. Google Cloud SDK installed and configured
2. Project created with billing enabled
3. Required APIs enabled (Cloud Run, Cloud Build)

Deploy

```
# Set your project ID
PROJECT_ID="your-project-id"

# Build and push image
gcloud builds submit --tag gcr.io/$PROJECT_ID/python-executor

# Deploy to Cloud Run
gcloud run deploy python-executor \
  --image gcr.io/$PROJECT_ID/python-executor \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --memory 1Gi \
  --cpu 1 \
  --timeout 60s
```

Architecture

NsJail Configuration

Cloud Run compatible configuration with filesystem isolation:

```
nsjail_cmd = [
    '/usr/local/bin/nsjail',
    '-Me',  # Execve mode
    '--time_limit', '30',  # Timeout enforcement
    '--disable_rlimits',  # Cloud Run compatibility

    # Filesystem restrictions (read-only bind mounts)
    '-R', '/usr/local/lib/python3.11',  # Python libs (read-only)
    '-R', '/usr/local/bin/python3.11',  # Python binary (read-only)
    '-R', '/lib',       # System libs (read-only)
    '-R', '/lib64',     # System libs (read-only)
    '-R', '/usr/lib',   # User libs (read-only)
    '--tmpfsmount', '/tmp',  # Only /tmp is writable

    '-E', 'PATH=/usr/local/bin:/usr/bin:/bin',

    # Namespace isolation (disabled for Cloud Run)
    '--disable_clone_newnet',
    '--disable_clone_newuser',
    '--disable_clone_newns',
    '--disable_clone_newcgroup',
    '--disable_clone_newipc',
    '--disable_clone_newuts',
    '--disable_clone_newpid',

    '--quiet',
    '--',
    '/usr/local/bin/python3.11',
    script_path
]
```

Key Features:

- `-R` flags create read-only bind mounts (works without chroot/elevated privileges)
- Customer code cannot access `/app/` directory (secrets protected)
- Only `/tmp` is writable via tmpfs
- All namespace clones disabled for Cloud Run compatibility
- Filesystem isolation without requiring CAP_SYS_CHROOT capability

Technical Stack

- **Framework**: Flask + Gunicorn
- **Sandbox**: NsJail (Cloud Run optimized)
- **Platform**: Google Cloud Run
- **Python**: 3.11
- **Libraries**: pandas, numpy, standard library

Project Structure

```
.
├── app.py              # Flask application with NsJail integration
├── Dockerfile          # Container definition with NsJail build
├── README.md          # This file
└── .gitignore         # Git ignore rules
```

Error Codes

- **200**: Successful execution
- **400**: Bad request (validation error, execution error)
- **404**: Endpoint not found
- **500**: Internal server error

Limitations

- Maximum execution time: 30 seconds
- Memory limit: 1GB (Cloud Run container level)
- No per-script memory limits (Cloud Run restriction)

- Limited to installed libraries (pandas, numpy, standard library)
- `/tmp` filesystem only (isolated from host)

Challenges Faced & Solutions

Challenge 1: NsJail Filesystem Isolation on Cloud Run

Problem: Cloud Run's unprivileged containers don't allow `chroot()` or mount namespace operations. Initial attempts with `--chroot '/'` failed with "Operation not permitted" errors.

Solution: Used read-only bind mounts (`-R` flags) instead of chroot. Mounted only essential directories (Python libs, system libs) as read-only, created writable `/tmp` via `--tmpfsmount` . This achieves filesystem isolation without requiring elevated capabilities. Customer code cannot access `/app/` directory where secrets live.

Challenge 2: Cloud Run RLIMIT_RTPRIO Restriction

Problem: NsJail's resource limit enforcement (`setrlimit` syscalls) is blocked by Cloud Run with "Operation not permitted" on `RLIMIT_RTPRIO` .

Solution: Added `--disable_rlimits` flag to skip rlimit enforcement. Relied on Cloud Run's platform-level limits (1GB RAM, 1 vCPU) and kept 30-second timeout for loop/crash protection. Trade-off: no per-script memory caps, but Cloud Run prevents total service crashes.

Challenge 3: NsJail Mode Selection

Problem: Different NsJail modes (`-Mo` , `-Me` , `-Ml`) have varying compatibility with Cloud Run. Standard once mode (`-Mo`) had namespace conflicts.

Solution: Switched to execve mode (`-Me`) which works without namespace isolation. Disabled all namespace clone flags (`--disable_clone_*`). NsJail provides supervision + filesystem isolation while Cloud Run provides actual process/VM isolation.

Challenge 4: Python Path Resolution

Problem: Container-built Python installs at `/usr/local/bin/python3.11` , not `/usr/bin/python3` .

Solution: Verified Python location with `which python3` in container. Updated all NsJail commands to use `/usr/local/bin/python3.11` and added correct path to `-R` bind mount flags.

Development Time

Approximate time to complete: **3 hours**
(Including NsJail Cloud Run compatibility research and testing)

Repository

GitHub: https://github.com/varadnair30/python-executor

Author

Varad Nair

- Email: vnairusa30@gmail.com
- Phone: +1 (657)-767-9035
- GitHub: github.com/varadnair30

License