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

- ✅ **Secure Execution**: NsJail sandboxing with Cloud Run's gVisor runtime
- ✅ **Resource Limits**: 30-second execution timeout, memory restrictions
- ✅ **Library Support**: Includes pandas, numpy
- ✅ **Input Validation**: Ensures scripts contain
- ✅ **JSON Response**: Structured output with results
- ✅ **Production Ready**: Deployed on Google

API Endpoint

POST /execute

Execute a Python script and return the result.

Request:

```
json

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

Success Response (200):

```
json
```

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

Error Response (400):

```
json

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

Requirements

- Script must contain a `def main()` function
- The `main()` function must return a JSON-serializable object
- Maximum execution time: 30 seconds
- Available libraries: os, pandas, numpy

Example Usage

Example 1: Simple Calculation

```
bash

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:

```
json

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

Example 2: Using Pandas

```bash
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\": [1, 2, 3], \"b\": [4, 5, 6]})\n    return {\"sum_a\"
  }'
```

Response:

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

Example 3: Using NumPy

```bash
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([1, 2, 3, 4, 5])\n    return {\"mean\": float(arr.mean()), \"std\
  }'
```

Response:

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

Example 4: Using OS Module

```bash
```

```
curl -X POST https://python-executor-wddqxxteba-uc.a.run.app/execute \\
  -H "Content-Type: application/json" \\
  -d '{
    "script": "import os\n\ndef main():\n    return {\"platform\": os.name, \"cpu_count\": os.cpu_count()}"
  }'
```

Response:

```
json

{
  "result": {"platform": "posix", "cpu_count": 2},
  "stdout": ""
}
```

Example 5: Error Handling

```
bash

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:

```
json

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

Local Development

Prerequisites

- Docker installed
- Google Cloud SDK (for deployment)

Run Locally

```bash
# 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

Deploy

```bash
# 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

Security

1. **NsJail Sandboxing**:
   - Runs in MODE_STANDALONE_EXECVE mode
   - All namespace isolation disabled (Cloud Run provides isolation)
   - 30-second execution timeout
   - Process supervision
2. **Cloud Run Security**:
   - gVisor runtime provides kernel-level isolation
   - Memory limits (1GB)
   - CPU throttling
   - Network isolation
3. **Input Validation**:
   - Checks for `main()` function presence
   - Validates JSON return type
   - Script size limit (100KB)

Technical Stack

- **Framework**: Flask + Gunicorn
- **Sandbox**: NsJail (minimal configuration for Cloud Run compatibility)
- **Platform**: Google Cloud Run
- **Python**: 3.11
- **Libraries**: pandas, numpy, os

Implementation Notes

NsJail Configuration

After extensive testing, the working NsJail configuration for Cloud Run is:

```python
```

```
nsjail_cmd = [
    '/usr/local/bin/nsjail',
    '-Me',  # MODE_STANDALONE_EXECVE
    '-t', '30',  # 30 second timeout
    '--disable_proc',
    '--disable_clone_newuser',
    '--disable_clone_newnet',
    '--disable_clone_newns',
    '--disable_clone_newpid',
    '--disable_clone_newipc',
    '--disable_clone_newuts',
    '--disable_clone_newcgroup',
    '--disable_rlimits',  # Required for Cloud Run
    '-q',
    '--',
    '/usr/local/bin/python3',
    script_path
]
```

Key Insights:

- Cloud Run's gVisor sandbox restricts certain Linux capabilities
- NsJail's execve mode (`-Me`) works where clone mode fails
- All namespace creation must be disabled
- `--disable_rlimits` is required to avoid `RLIMIT_RTPRIO` permission errors
- Cloud Run's own isolation provides the primary security layer

Project Structure

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

Error Codes

- **200**: Successful execution

- **400**: Bad request (validation error, execution error)
- **500**: Internal server error

Limitations

- Maximum execution time: 30 seconds
- Memory limit: 1GB (Cloud Run configuration)
- No internet access during execution
- Limited to pandas, numpy, and os libraries
- Cannot write persistent files

Repository

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

Challenges Faced & Solutions

Challenge 1: NsJail Incompatibility with Cloud Run's gVisor Runtime

Problem: NsJail's default configuration requires Linux capabilities (`PR_CAP_AMBIENT`, `RLIMIT_RTPRIO`) that are restricted in Cloud Run's gVisor sandbox. Initial attempts with standard NsJail configurations consistently failed with "Operation not permitted" errors.

Solution:

- Switched from clone-based mode (`-Mo`) to execve mode (`-Me`)
- Disabled all namespace creation flags (`--disable_clone_*`)
- Added `--disable_rlimits` to prevent capability conflicts
- Used `/usr/local/bin/python3` (correct container path)
- Result: NsJail runs in minimal supervision mode while Cloud Run's gVisor provides actual isolation

Challenge 2: Docker Image Caching Issues

Problem: During iterative testing, Docker was caching old versions of `app.py`, causing updated NsJail configurations not to be reflected in deployed containers.

Solution:

- Modified Dockerfile to use `COPY . .` instead of `COPY app.py .`
- Incremented version tags for each build (v1, v2, ... v22)

- Always ran both `gcloud builds submit` AND `gcloud run deploy` commands
- Added cache-busting `ARG CACHEBUST=1` in Dockerfile

Challenge 3: Python Path Resolution in Container

Problem: NsJail couldn't find Python at `/usr/bin/python3` because the Docker image installs it at `/usr/local/bin/python3`.

Solution:

- Used `grep` to verify Python installation location in container
- Updated NsJail command to use correct path
- Verified with `which python3` during container builds

Challenge 4: Understanding NsJail Flag Syntax

Problem: NsJail documentation shows protobuf configs, but command-line flags work differently. Wrong flag syntax (`--mode o` vs `-Mo`) caused parsing errors.

Solution:

- Studied NsJail's help output (`nsjail --help`)
- Used combined short flags (`-Mo`, `-Me`) instead of long form
- Referenced Windmill's production config for real-world examples
- Tested flag combinations incrementally

Development Time

Approximate time to complete: **2.5 hours**
(Including debugging NsJail compatibility: ~1.5 hours)

Author

Varad Nair

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

License

MIT License