A Python Sandbox for Dynamic Rule Execution

Problems and Practices

Zhiya

Agenda

- What is it
- Why and Why Not
  - VM vs Docker
  - Python Sandbox
- Practices
  - Isolation
  - Security
  - Performance

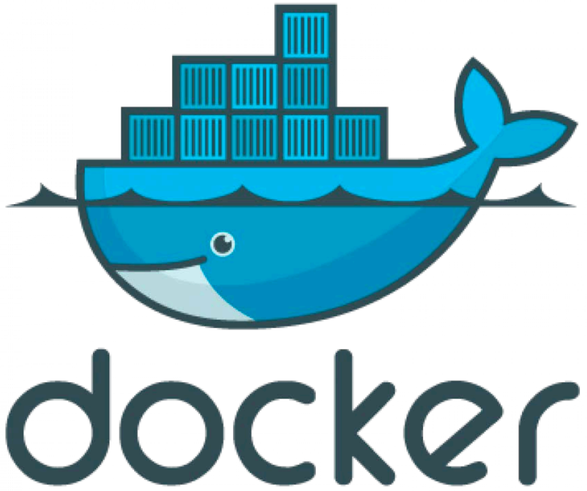
What is it

Context

- Rule
  - A set of expressions that express business logics
  - Expressions are in Python
- Rule Engine
  - Executing all the rules
  - No isolation, No control cause internal/external risks

Why and Why Not

VM vs Docker



Why and Why Not

Python Sandbox

- Isolation
  - Restricts access to sensitive resources (hardware, network, system calls)
  - Customization options for resource limits and security policies
- Security
  - Prevent malicious code
  - Blocklist for control over python modules and functions
  - Error Handling to prevent system crashes
- Performance
  - Massive Requests
  - Low Latency

Practices

Isolation

- Resource Usage Limitaion
- Case Level Dynamic Control
- Running Time Control

`resource` — Resource usage information

This module provides basic mechanisms for measuring and controlling system resources utilized by a program.

Availability: Unix, not Emscripten, not WASI.

Symbolic constants are used to specify particular system resources and to request usage information about either the current process or its children.

An `OSError` is raised on syscall failure.

exception resource.**error**

A deprecated alias of `OSError`.

Changed in version 3.3: Following PEP 3151, this class was made an alias of `OSError`.

```python
usage = resource.getrusage(resource.RUSAGE_SELF)

# set maximum cpu time
current_cpu = math.ceil(usage.ru_utime + usage.ru_stime)
cpu = current_cpu + math.ceil(max_cpu / 1000.0)
resource.setrlimit(resource.RLIMIT_CPU,
                   (cpu, resource.RLIM_INFINITY))

# set maximum memory usage
current_mem = math.ceil(usage.ru_maxrss)
mem = current_mem + math.ceil(max_mem)
resource.setrlimit(resource.RLIMIT_AS,
                   (mem, resource.RLIM_INFINITY))
current_stack = math.ceil(usage.ru_isrss)

# set maximum stack usage
stack = current_stack + math.ceil(max_stack)
resource.setrlimit(resource.RLIMIT_STACK,
                   (stack, resource.RLIM_INFINITY))

# set maximum number of processes
resource.setrlimit(resource.RLIMIT_NPROC,
                   (max_proc, resource.RLIM_INFINITY))

# set maximum file size
resource.setrlimit(resource.RLIMIT_FSIZE,
                   (max_file_size, resource.RLIM_INFINITY))
```

Practices

Security - Malicious Code Prevention

- Blocklist + AST Analysis
- Module Level / Function Level

`ast` — Abstract Syntax Trees

Source code: Lib/ast.py

The `ast` module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like.

An abstract syntax tree can be generated by passing `ast.PyCF_ONLY_AST` as a flag to the `compile()` built-in function, or using the `parse()` helper provided in this module. The result will be a tree of objects whose classes all inherit from `ast.AST`. An abstract syntax tree can be compiled into a Python code object using the built-in `compile()` function.

Practices

Security

- Error Handling
- Traceback Printing

```python
context = {}
code = """
import math

a = "abc"
print(math.sqrt(a))
"""

try:
    exec(code, context)
except Exception as e:
    stderr = traceback.format_exc()
    print(stderr)
```

```
Traceback (most recent call last):
  File "/Users/zzy/Desktop/test/test_audit.py", line 16, in <module>
    exec(code, context)
  File "<string>", line 5, in <module>
TypeError: must be real number, not str
```

Practices

Performance

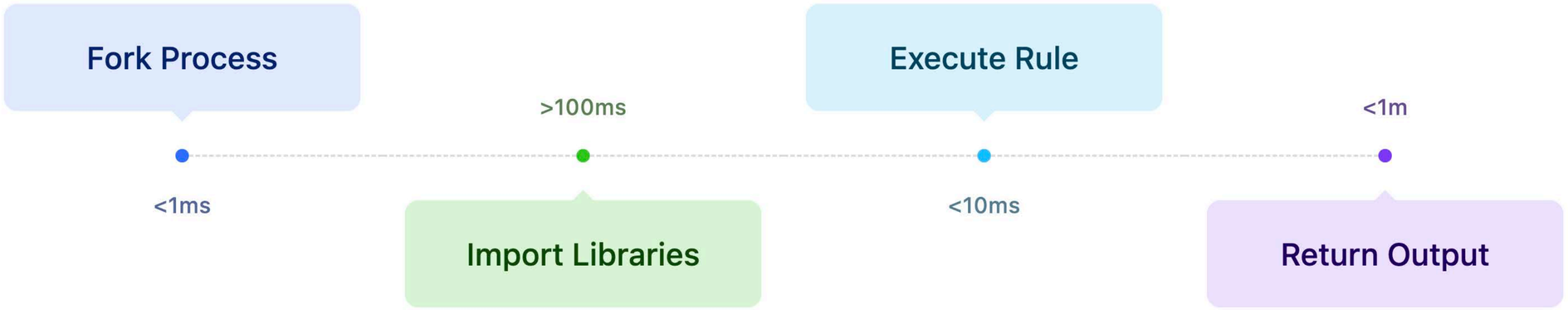
- Reuse of Sandbox
- Dependencies Preloading

import requests
16 Processes
8 Core CPU
SSD

```
python3 main.py | sort
0.20s
0.21s
0.22s
0.24s
0.24s
0.25s
0.26s
0.26s
0.26s
0.27s
0.27s
0.27s
0.28s
0.30s
0.31s
0.31s
```

```python
context = {}
code = """
import requests
"""

try:
    start = time.time()
    exec(code, context)
    duration = time.time() - start
    print(f"{format(duration, '.2f')}s")
except Exception as e:
    stderr = traceback.format_exc()
```



Thanks