Flask-Faasm: Deploying Serverless Web APIs on *Faasm*

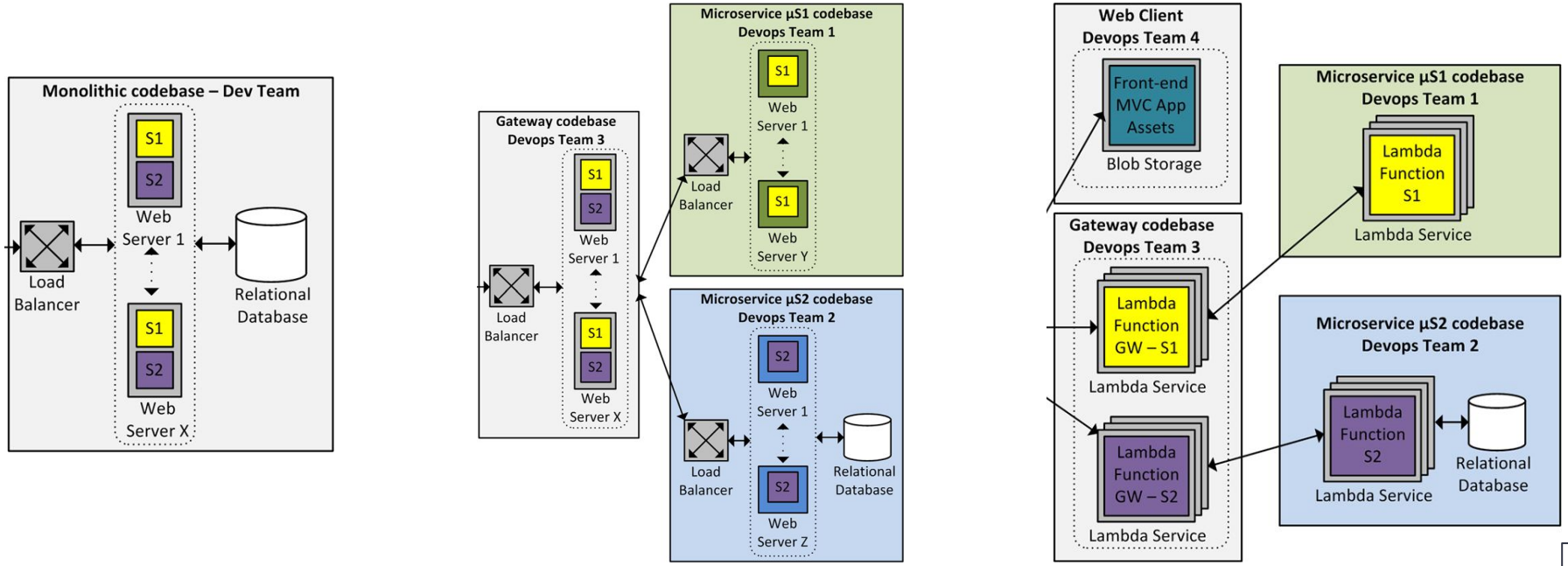
Hyunhoi Koo

June 27th, 2023

Contents

The Move to the Serverless Paradigm

Monolithic → **Microservice** → **Serverless**



[1]

Serverless and Web API

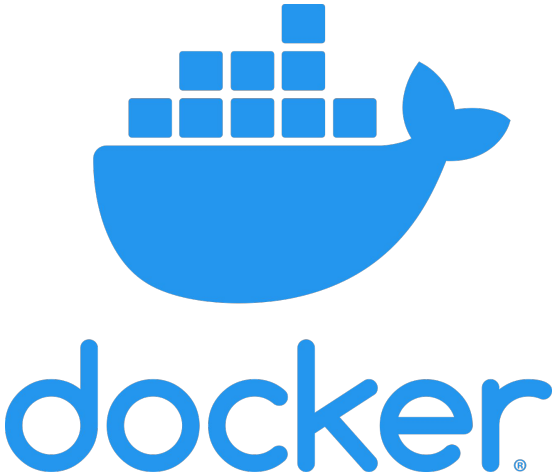
Developer Convenience

‘Infinite’ Scaling

Pricing

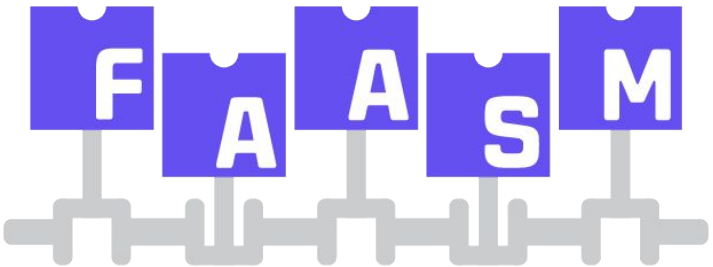
FaaS: Function-as-a-Service

Problems with Serverless



[2]

Faaslets and Faasm



- Faaslets
  - WebAssembly
  - Software-Fault Isolation (SFI)
  - Linux cgroups
- Proto-Faaslets
- Faasm

[3]

Motivation

Is it **viable** to deploy Web API on alternative Serverless FaaS Platforms like **Faasm** if it uses **Software-Based Isolation Mechanisms** for resource management?

Develop a product that enables an easy deployment of Web API on Faasm

System Design

Design Requirements

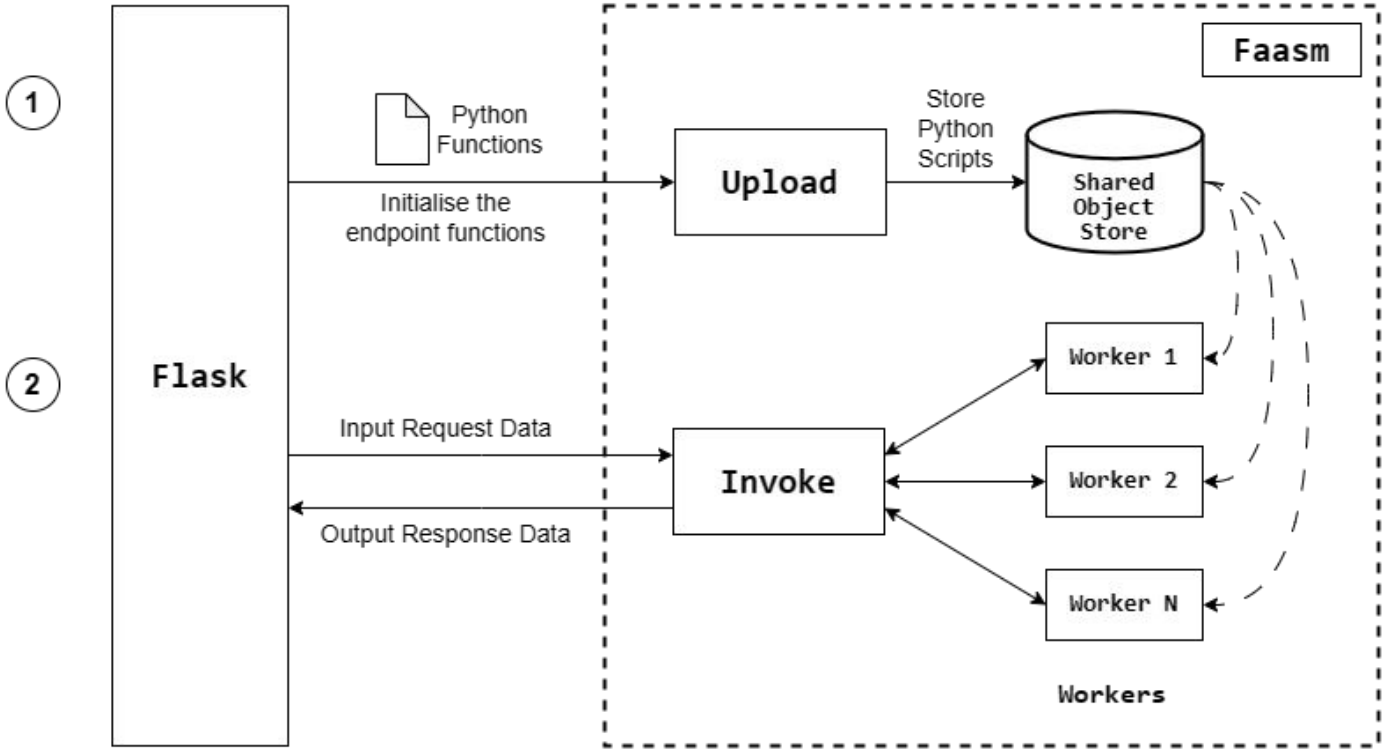
Requirements

- Python
- Flask
- ‘Zero-cost’
- The Critical Loop

Limitations

- HTTP
- Polling-based async
- Python in Faasm
- Third-party Libraries

System Architecture Blueprint



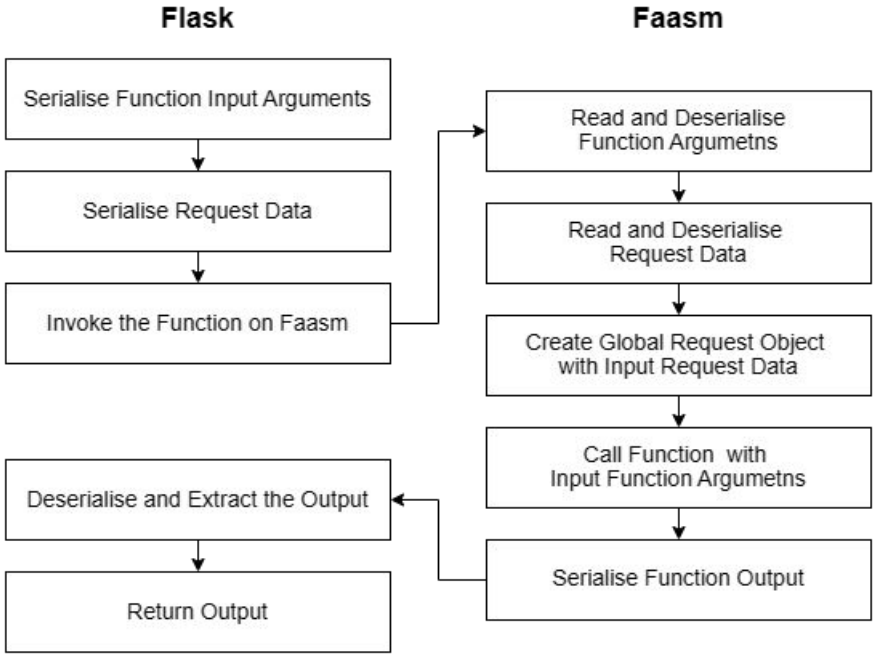
Flask-Faasm

Workflow of *Flask-Faasm*

Upload Phase

1. **Package endpoint functions** to be **compatible** with Faasm
2. **Upload** the packaged function to Faasm
3. **Replace** the endpoint function in the Flask app with a function that **invokes the uploaded function** on Faasm

Invoke Phase



Implementation Details

Q. How are Python functions executed on Faasm?

Q. How can external systems upload and invoke functions on Faasm?

Q. How can the behaviour of Flask's endpoint functions be changed without modifying the app itself?

Q. How does Flask-Faasm create an interface between Flask and Faasm?

Executing Python Functions on Faasm

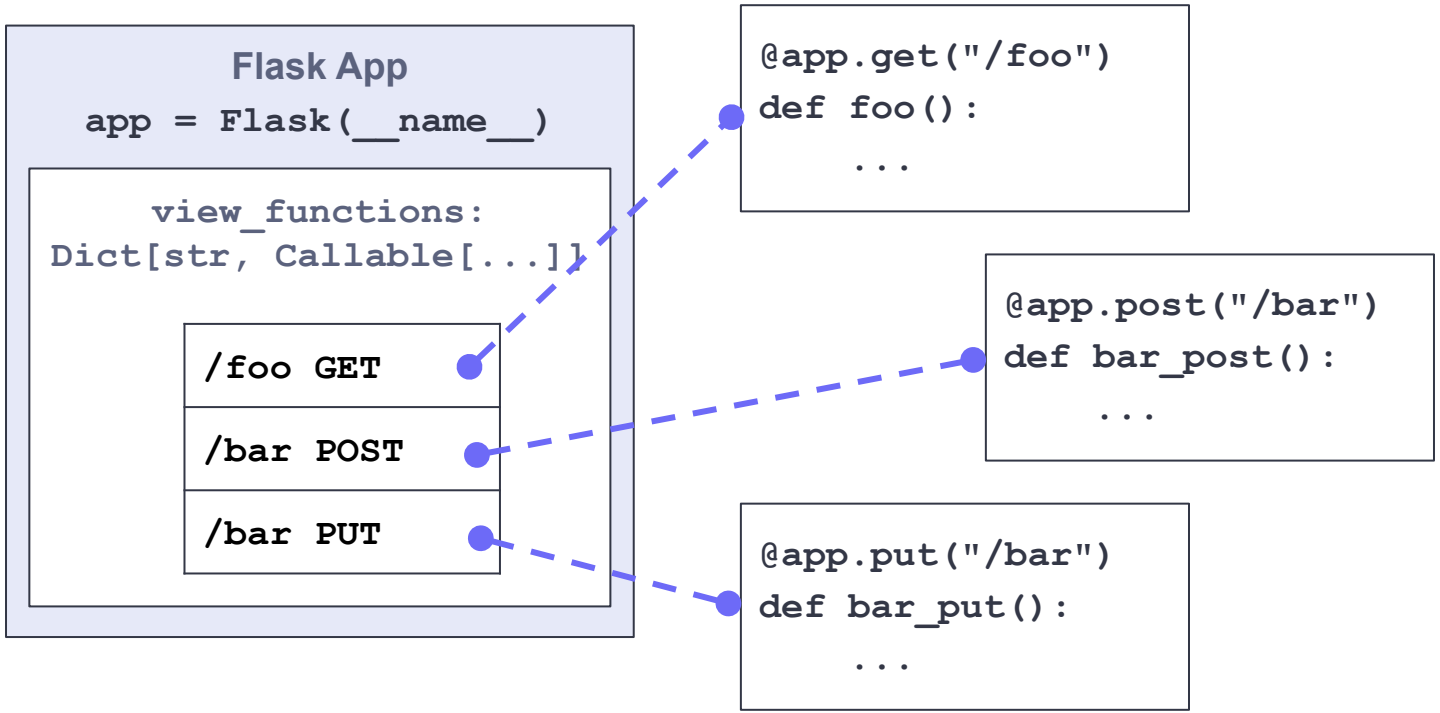
```python
from pyfaasm.core import get_input_len, read_input, write_output

def echo() -> None:
    input_len = get_input_len()
    if input_len == 0:
        write_output("Nothing to echo")
        return

    input_data: str = read_input(input_len).decode("utf-8")

    write_output(input_data_str.encode(encoding="utf-8"))

def faasm_main() -> int:
    echo()
    return 0
```

Faasm HTTP API: Upload & Invoke Services

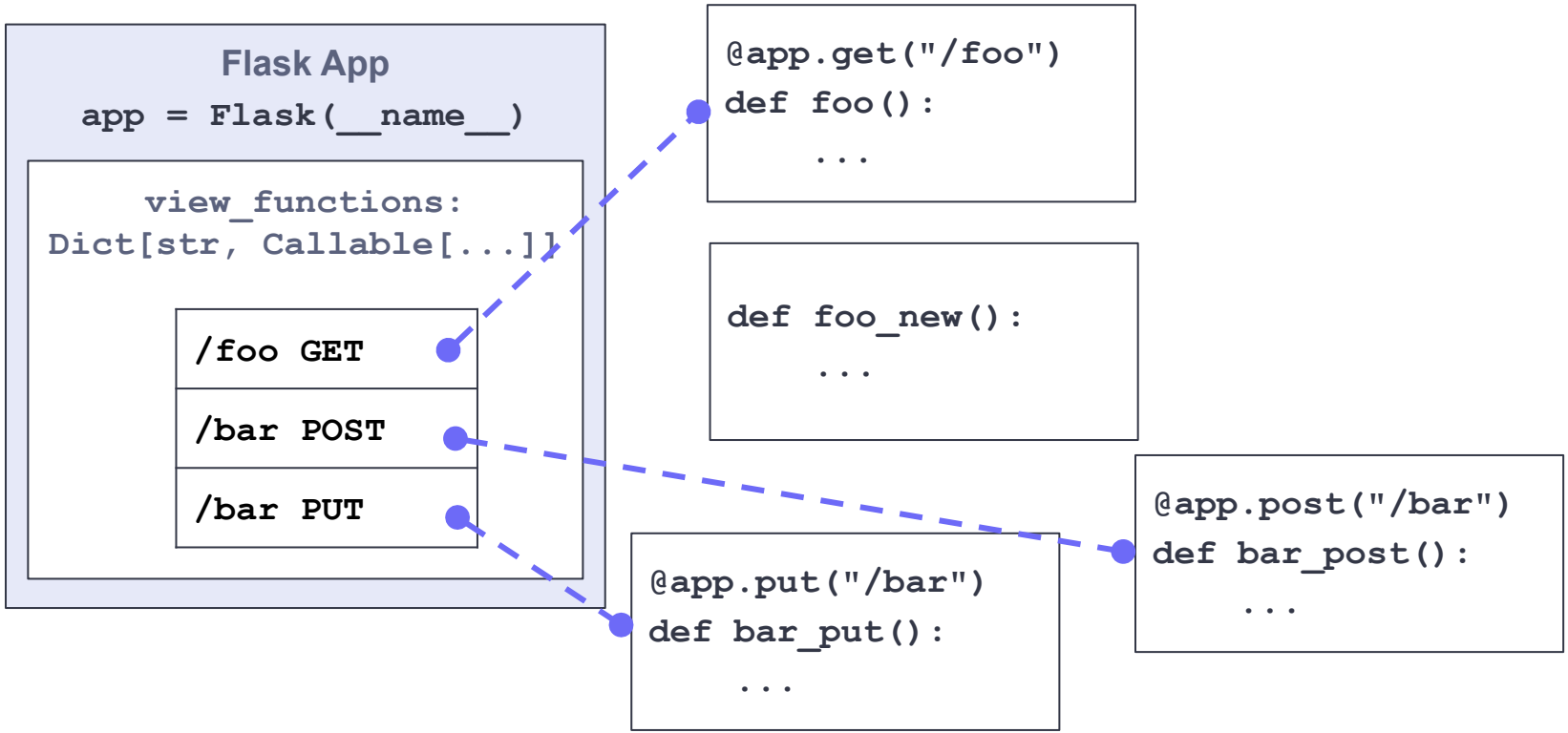
- URI: **`/<user>/<function>`**
- C/C++: **`/f/<user>/<function>`**
  - e.g. CPython Runtime @ **`/f/python/py_func`**
- Python: **`/p/<user>/<function>`**

```
1 <output on stdout / stderr>
2 Python call succeeded
3
4
5 <output from write_output() function>
```

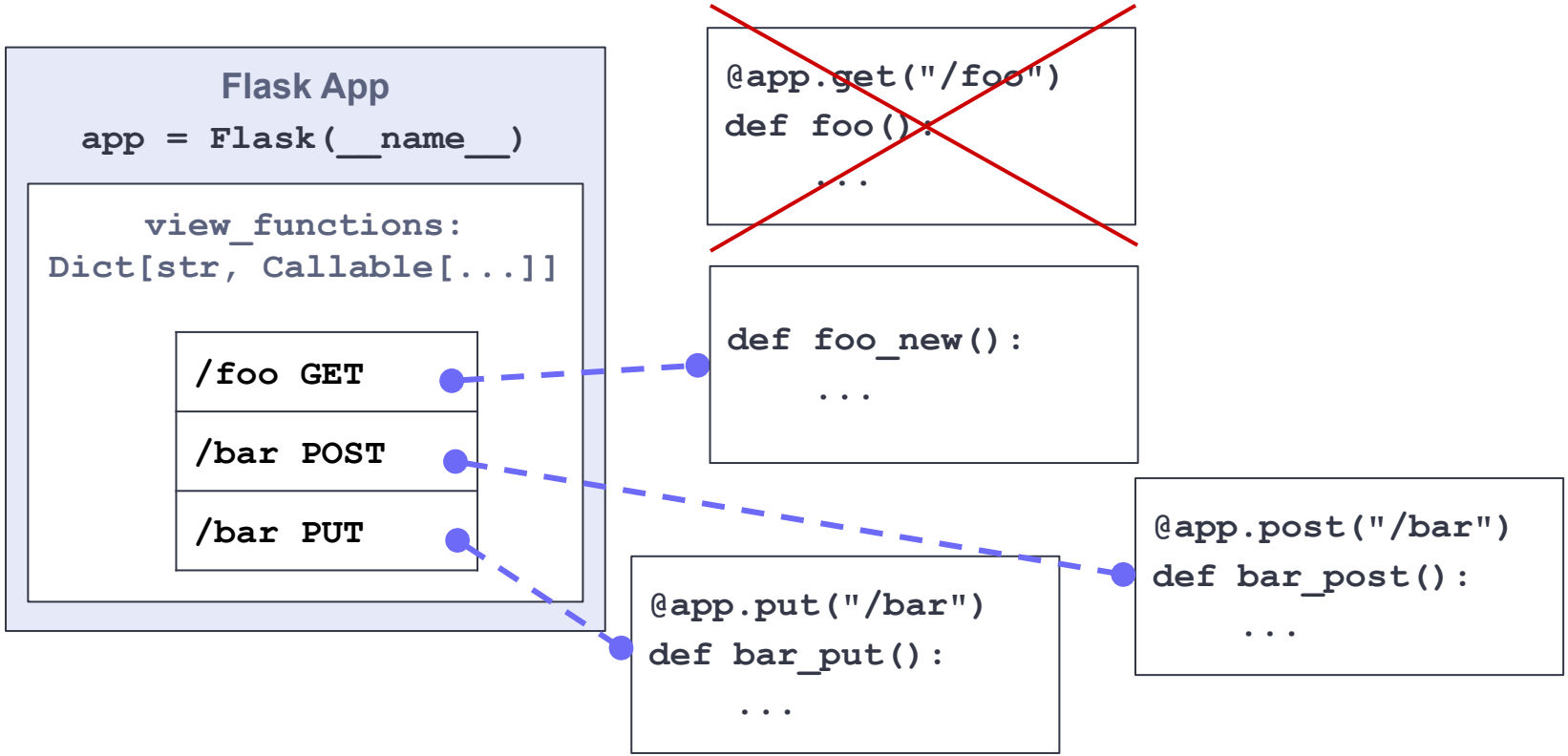
‘Hijacking’ HTTP Requests to Flask Apps



‘Hijacking’ HTTP Requests to Flask Apps



‘Hijacking’ HTTP Requests to Flask Apps



Interfacing Flask and Faasm

Function Template

```
<imports>

<request proxy object>

<module library>

<function>

def faasm_main() -> int:
    # Read input data and parse as JSON.
    # Extract args, kwargs, and request data.
    # Call the function.
    # Encode the function output as JSON
      and write to output.
```

- Hijack endpoint functions on Flask & Invoke the uploaded function on Faasm
- Source code for each function + other information inserted into template
- Resulting Function Script uploaded to Faasm
- Additional Information:
  - Function I/O
  - Request Object
  - Imports
  - Module Library

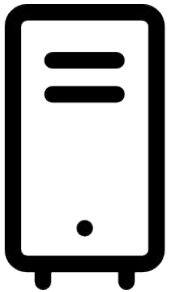
Demo

Evaluation

Experimental Setup

DoC Research Cluster

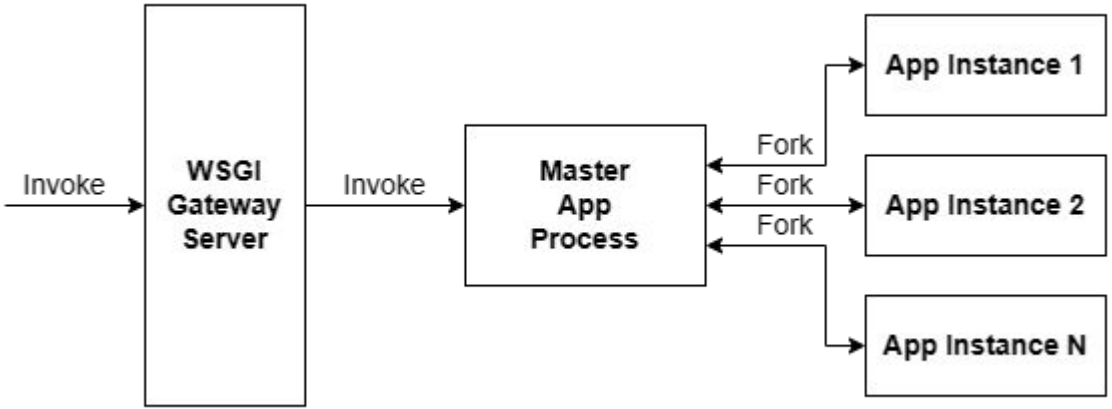
Benchmark Web API



Faasm Instance

Flask App Server

Baseline ‘Native’ Setup



Benchmark Web API

- **Python Performance Benchmark Suite** (PyPerformance)
  - Low-level micro-benchmark functions + real-world applications
- Standard benchmark suite for Python-related systems
  - Used in various research papers
  - Including the original Faasm Paper
- Reliable and consistent set of performance metrics
- Subset of PyPerformance suite adapted as Web API

Experiment 1: Comparing the Latency of *Flask-Faasm* vs. ‘Native’ Deployment

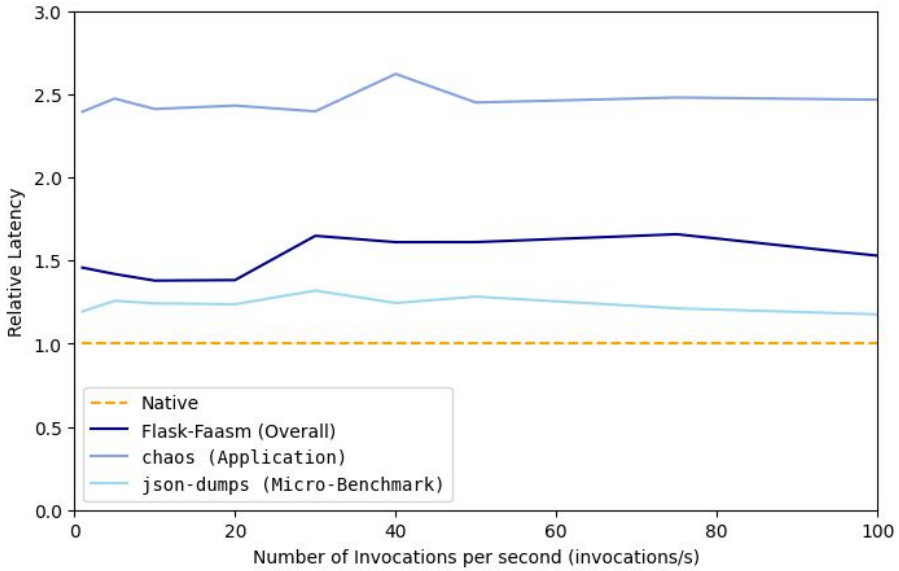
Motivation

- Inherent overhead in *Flask-Faasm* vs Native Baseline
  - *Flask-Faasm* ⊃ Native
- Overhead for:
  - Latency in communication between Flask app and Faasm
  - Serialising and deserialising input/output
  - Function script loading to WebAssembly CPython runtime
- If less than an order of magnitude (or above) overhead:
  - Considerable improvement over existing systems
  - Hardware-based isolation mechanisms

Methodology

```
foreach function in benchmark API:

    foreach invocation_amount in [1, 5, 10, 20, 30, 40, 50, 75, 100]:

        capture latency of invoking function in invocation_amount
        concurrent requests 5 times

        obtain median latency for the 5 invocations

repeat the above for both native and Flask-Faasm invocations

Calculate the relative latency overhead of Flask-Faasm compared to
native execution

Take arithmetic mean of relative latency across all benchmark
functions
```

Results: Average Relative Latency Overhead



- 1.5x latency over native
  - Significantly better than traditional serverless deployment
- Micro-benchmark vs Application
  - json-dumps: 1.2-1.3x
  - chaos: 2.5x
- Web API: ephemeral functions
- Still a considerable overhead
  - Idealised scenario

Experiment 2: Examining the Source of Overheads in *Flask-Faasm* Function Execution

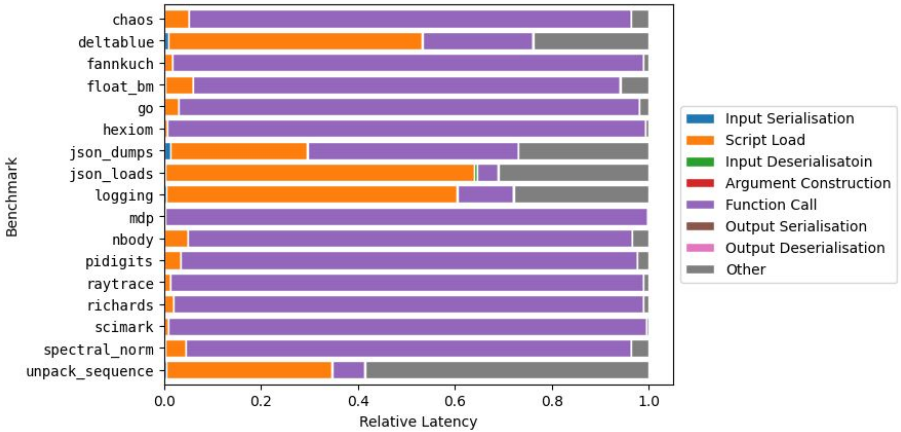
Motivation

- Still a considerable overhead over native execution
  - I.e. vs Monolithic deployment
- Quantify and analyse the main contributors to the latency overhead

Methodology

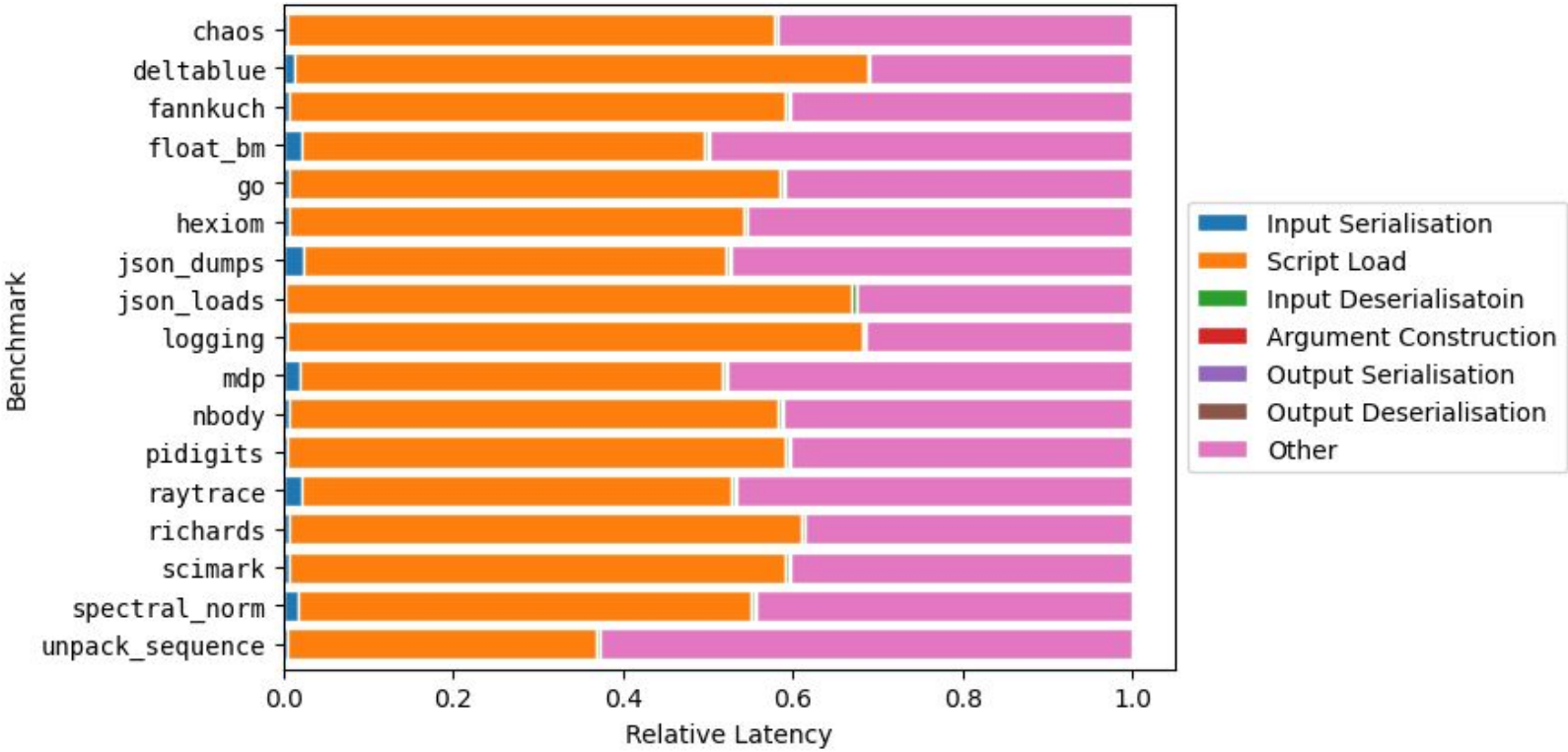
- Each benchmark functions run 10 times on *Flask-Faasm*
- Collect the latency measurements
  - Calls to **`perf_counter()`** in various locations of the script
  - e.g. at the beginning and end of each script → script load time
- Measurements collected → arithmetic mean

Results: Relative Latency Distribution



- Script Load time dominates additional latency
- Input and Output (de)serialisation trivial
- Latency overhead more significant for ephemeral functions

Results: Without Function Execution



Conclusion

- *Flask-Faasm*: Zero-cost adapter interface between Flask Web API and Faasm
- Latency overhead: around 1.5x native baseline
  - Smaller than serverless platforms that use hardware-based isolation mechanisms
- Latency profile of function invocation on Flask-Faasm
  - ‘Script load’ time most significant

Future Work

1. Introduce Proto-Faaslets for Pre-Initialising Functions
2. A New Serialisation Method
3. Deploy Flask-Faasm on IaaS Platforms
4. Testing Flask-Faasm with a Commercial-Level Web API

Questions?

Bibliography

[1] Villamizar M, Garcés O, Ochoa L, Castro H, Salamanca L, Verano M, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. Service Oriented Computing and Applications. 2017;11(2):233-47.

[2] https://www.docker.com/company/newsroom/media-resources/

[3] https://github.com/faasm/faasm

[4] https://flask.palletsprojects.com/en/2.3.x/