FaaSLight Serverless Simulation User Guide

Mohab

May 27, 2025

Contents

1 Introduction

FaaSLight is a Python-based project that simulates and compares serverless computing models, evaluating five platforms: FaaSLight Original, FaaSLight Enhanced, SAND++, AWS Fargate, and Unikernel. It measures performance across compute, I/O, and memory-intensive workloads, focusing on latency, CPU/memory usage, security, cost, and a composite score. The project uses Docker containers to host Flask APIs, a Python simulator to generate workloads, and Plotly for interactive visualizations. This guide provides instructions for setup, usage, and details on the project's architecture, functions, and execution flow.

2 Prerequisites

- **Operating System**: Windows (tested), Linux, or macOS.
- **Docker Desktop**: Latest version.
- **Python**: 3.10 or higher.
- **Disk Space**: Approximately 500MB for Docker images and results.
- **RAM**: 8GB recommended (4GB minimum).
- **CPU**: 4 cores recommended.

3 Installation

1. **Clone the Repository**:

```
git clone https://github.com/<your-username>/faaslight.git
cd faaslight

```

2. **Set Up Python Virtual Environment**:

```
python -m venv venv
source venv/Scripts/activate  % Windows
% or source venv/bin/activate  % Linux/macOS
pip install -r requirements.txt

```

3. **Start Docker Desktop**: Ensure Docker is running.

4 Usage

1. **Start Docker Containers**:

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

```

Verify containers are running:

```
docker-compose ps

```

Expected output:

```
Name                                          Command               State
------------------------------------------------------------------------------
faaslight_project_new-faaslight_enhanced-1   flask run --host=0.0.0.0   Up
faaslight_project_new-faaslight_original-1   flask run --host=0.0.0.0   Up
...
```

2. **Run the Simulation**:

```
python faaslight_simulator_.py
```

The simulation takes approximately 1–2 minutes, prints performance metrics, and generates `results/metrics.csv` and `results/dashboard.html`.

3. **View Results**: Open `results/dashboard.html` in a web browser to view interactive bar and scatter plots.

4. **Clean Up**:

```
docker-compose down
deactivate
```

5 Architecture

The FaaSLight project simulates serverless platforms using a Docker-based architecture:

- **Docker Containers**: Five services (`faaslight_original`, `faaslight_enhanced`, `sand_plus`, `fargate_inspired`, `unikernel_inspired`) are defined in `docker-compose.yml` and built from `Dockerfile` using Python 3.9-slim.
- **Flask APIs**: The `app.py` script runs a Flask server in each container, providing endpoints (`/compute`, `/io`, `/memory`, `/verify`) to simulate serverless workloads.
- **Simulator**: The `faaslight_simulator_.py` script sends HTTP requests to the containers, collects performance metrics, and generates visualizations using Plotly.
- **Network**: A bridge network (`faaslight-net`) connects containers to the host, mapping container port 5000 to localhost ports 32768–32772.
- **Output**: Results are saved as `results/metrics.csv` (raw metrics) and `results/dashboard.htm` (interactive plots).

6 Key Functions

The following are the primary functions in `faaslight_simulator_.py` and `app.py`:

6.1 `faaslight_simulator_.py`

- **`send_request(url, func_type, is_cold=False)`**:

  - *Purpose*: Sends an HTTP GET request to a service endpoint (e.g., `http://localhost:32769`).
  - *Parameters*:
    * `url`: Service URL (e.g., `http://localhost:32769`).
    * `func_type`: Workload type (`compute`, `io`, `memory`).
    * `is_cold`: If `True`, applies a cold-start penalty (e.g., 10ms for FaaSLight Enhanced).
  - *Returns*: Dictionary containing `latency` (ms), `status` (HTTP code), `type`, `security_overhead` (ms), `cpu_usage` (%), `memory_usage` (%).
  - *Example*:

```
result = send_request("http://localhost:32769", "compute", is_cold=True)
# Returns: {"latency": 750, "status": 200, "type": "compute", "security_overhead": 10, ...}

```

- **`run_simulation()`**:
  - *Purpose*: Executes the simulation for all services, sending 200 requests per service with 10% cold starts.
  - *Logic*: Uses `ThreadPoolExecutor` with 10 workers to parallelize requests. Collects metrics and performs security verification.
  - *Returns*: Dictionary of raw metrics per service and workload.
- **`compute_metrics(metrics)`**:
  - *Purpose*: Processes raw metrics into average and P95 latency, CPU/memory usage, security score, cost, and composite score.
  - *Formulas*:
    * Cost: $(avg\_cpu + avg\_memory) \times avg\_latency/1000$, reduced by 0.8 for FaaSLight Enhanced.
    * Composite: $(security \times 1000)/(avg\_latency + p95\_latency + cost)$.
  - *Returns*: Dictionary of processed metrics.
- **`plot_results(results)`**:
  - *Purpose*: Generates interactive visualizations using Plotly: a bar chart of composite scores and a scatter plot of latency vs. security.
  - *Output*: Saves `results/metrics.csv` and `results/dashboard.html`.

6.2 app.py

- **`compute()`**: Flask route simulating CPU-intensive tasks (e.g., matrix operations with NumPy).
- **`io()`**: Flask route simulating network-bound tasks (e.g., HTTP requests).

- `memory()`: Flask route simulating memory-intensive tasks (e.g., large array allocation).
- `verify()`: Flask route simulating a security check (e.g., SHA-256 hash), returning HTTP 200.

7 Calling Sequence

The execution flow of `faaslight_simulator_.py` is as follows:

1. **Main Block**: Initiates the simulation, printing the start timestamp.
2. `run_simulation()`:
   - Iterates over services (e.g., FaaSLight Enhanced).
   - Sends 200 requests per service using `send_request()`, with 20 cold starts.
   - Sends a `/verify` request for security checks.
3. `compute_metrics()`: Processes raw metrics into performance indicators.
4. `plot_results()`: Saves results to `metrics.csv` and generates `dashboard.html`.
5. **Completion**: Prints a message indicating output file locations.

8 Example Output

Below is an example output from a simulation run (single replica, 2025-05-27 13:06:06):

```
Starting FaaSLight simulation at 2025-05-27 13:06:06
Testing FaaSLight Enhanced...
FaaSLight Enhanced (compute): Avg Latency ~749ms, P95 Latency ~1488ms, CPU ~13.5%, Me
FaaSLight Enhanced (io): Avg Latency ~564ms, P95 Latency ~1206ms, CPU ~12.2%, Memory
FaaSLight Enhanced (memory): Avg Latency ~1095ms, P95 Latency ~1602ms, CPU ~12.1%, Me
...
Simulation complete. Results saved to results/dashboard.html and results/metrics.csv
```

With two replicas (recommended):

```
FaaSLight Enhanced (compute): Avg Latency ~150ms, P95 Latency ~180ms, CPU ~20.0%, Mem
FaaSLight Enhanced (io): Avg Latency ~50ms, P95 Latency ~70ms, CPU ~10.0%, Memory ~15
```

9 Troubleshooting

- **Containers Not Running**:

```
docker-compose logs faaslight_project_new-faaslight_enhanced-1

```

- **Port Conflicts**:

```
netstat -aon | findstr "32768 32769"
taskkill /PID <pid> /F

```

- **High Latency**: Switch to two replicas (see Section 9).
- **Dependency Issues**:

```
pip install -r requirements.txt

```

10 Two-Replica Setup for Improved Performance

To achieve lower latencies (150ms compute, 4.20 I/O composite score), configure two replicas for `faaslight_enhanced`:

1. **Edit `docker-compose.yml`**:

```
faaslight_enhanced:
  ...
  expose:
    - "5000"
  deploy:
    replicas: 2
    resources:
      limits:
        cpus: '0.75'
        memory: '768M'

```

   Remove `container_name` and `ports: ["32769:5000"]`.

2. **Edit `faaslight_simulator_.py`**:

```
def get_service_ports(service_name, num_replicas=1, project_prefix="faaslight_project_new"):
    import subprocess
    ports = []
    for i in range(1, num_replicas + 1):
        try:
            container_name = f"{project_prefix}-{service_name}-{i}"
            cmd = f"docker-compose port {container_name} 5000"
            output = subprocess.check_output(cmd, shell=True, text=True).strip()
            if output:
                host_port = output.split(":")[1]
                ports.append(f"http://localhost:{host_port}")
        except subprocess.CalledProcessError as e:
            print(f"Error getting port for {container_name}: {e}")
    return ports

services["FaaSLight Enhanced"] = {"url": None, "cold_start_penalty": 10, "scaling_factor": 2}
services["FaaSLight Enhanced"]["urls"] = get_service_ports("faaslight_enhanced", num_replicas=2)
if not services["FaaSLight Enhanced"]["urls"]:
    raise RuntimeError("No ports found for faaslight_enhanced replicas")

```

   Update `run_simulation()` to use `urls` (see repository).

3. **Ensure Resources**: Configure Docker Desktop with 4 CPUs, 8GB (Settings > Resources).
4. **Run**:

```
docker-compose up -d --build
python faaslight_simulator_.py

```