Frontend Development of Distributed FPGA Management in Serverless Computing

Zirong Cai
Advisor: Dr. Atsushi Koshiba, Jiyang Chen
Chair of Distributed Systems and Operating Systems
https://dse.in.tum.de/

15.10.2022 – 15.04.2023

Research context

- Function-as-a-Service (FaaS) is a popular way to deploy cloud services
  - Easy deployment, great scalability, low cost
- FPGA deliver increased performance at a lower cost compared with CPUs
  - Faster and more efficient processing for applications

Motivation

- Current FaaS frameworks do not have FPGA acceleration support
  - AWS Lambda, Azure functions
  - We can only use CPUs (and GPUs) to execute functions
  - Most cloud workloads however need FPGA accelerators

Urgent need to incorporate FPGA support within a serverless framework

Related Work

- BlastFunction [DATE’20] [ACM Trans’22][1]
  - Supports Time sharing, OpenCL programming
  - Implemented using OpenCL library , OpenFaaS, and Kubernetes
  - Does not support space sharing, no reconfiguration optimization
- Molecule [ASPLOS’22][2]
  - Supports Time sharing, optimized reconfiguration
  - Weak isolation between FPGA functions
- Using FPGAs as microservices[BPOE-9][3]
  - Exposes a microservice for each accelerator
  - Not a general API
  - Dose not support customized function

[1] A. Damiani, G. Fiscaletti, M. Bacis, R. Brondolin, and M. D. Santambrogio. "BlastFunction: A Full-Stack Framework Bringing FPGA Hardware Acceleration to Cloud-Native Applications."
[2] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2022. Serverless computing on heterogeneous computers.
[3] S. Ojika, A. Gordon-Ross, H. Lam, B. Patel, G. Kaul, and J. Strayer. "Using FPGAs as Microservices: Technology, Challenges and Case Study."

Problem statement

How to create a serverless architecture that incorporates FPGAs into the serverless execution paradigm?

Outline

- ~~Motivation~~

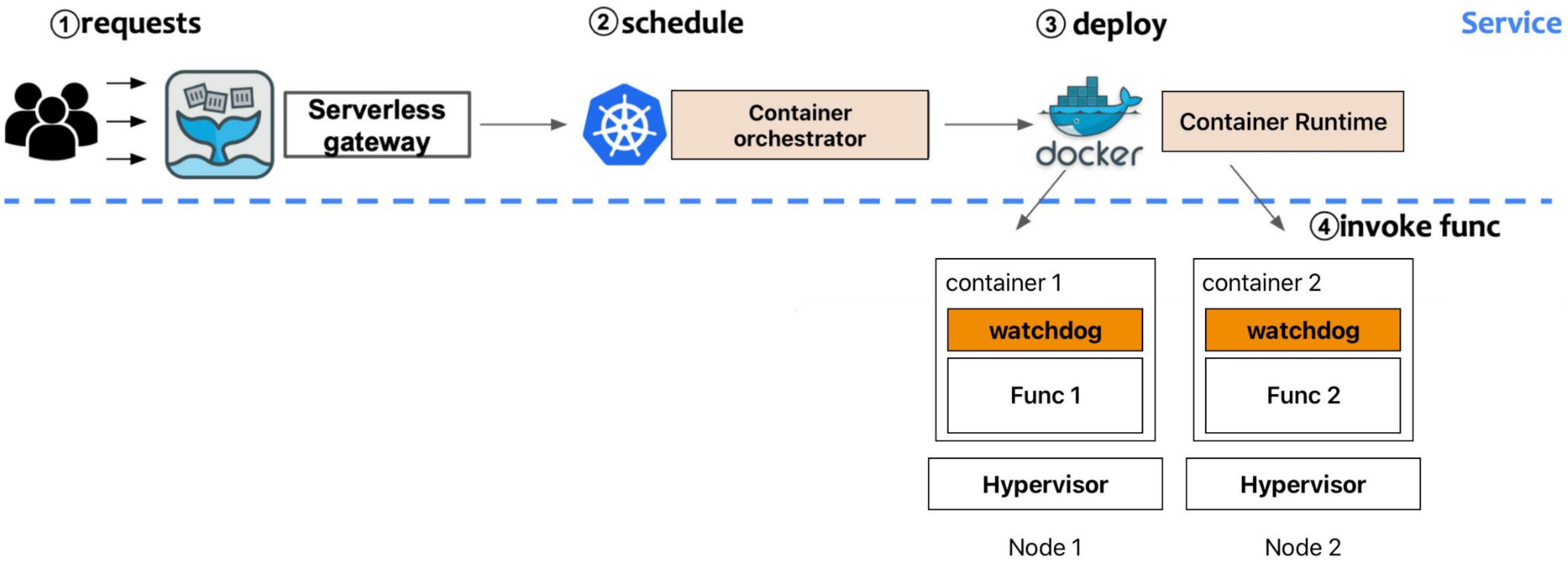
- Background
- Design

- Evaluation

- Summary

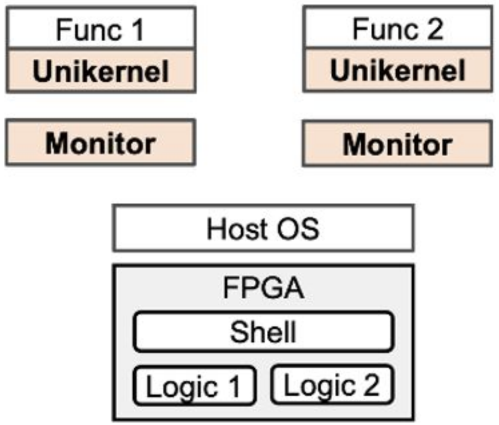
Background

- CPU-Centric serverless framework



Background

- Funky monitor

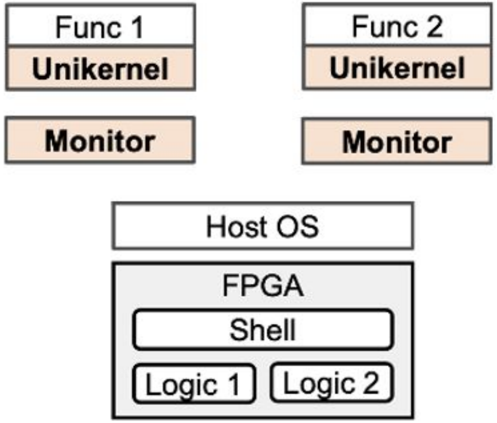


Funky application

Background

- Funky monitor
- Urunc

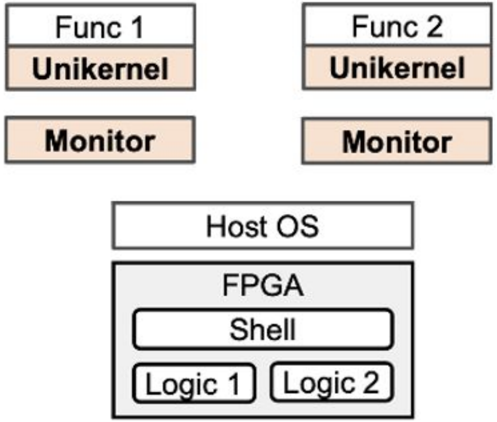
urunc



Funky application

Background

- Funky monitor
- Urunc
- Kubernetes
- OpenFaaS



Funky application

Outline

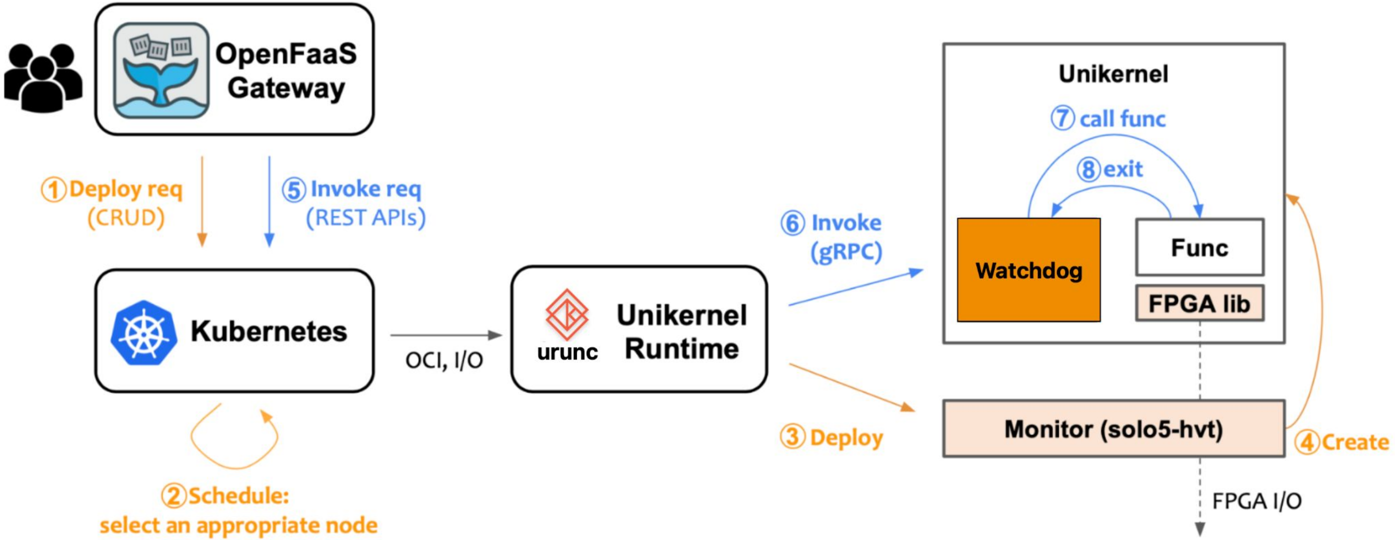
- ~~Motivation~~
- ~~Background~~
- Design
- Evaluation
- Summary

Design Challenge

- Compatibility with existing serverless framework
    - **Solution:** extend urunc to support Funky unikernel
- The unikernel function should be event-driven
    - **Solution:** Watchdog need to be integrated into Funky unikernel application

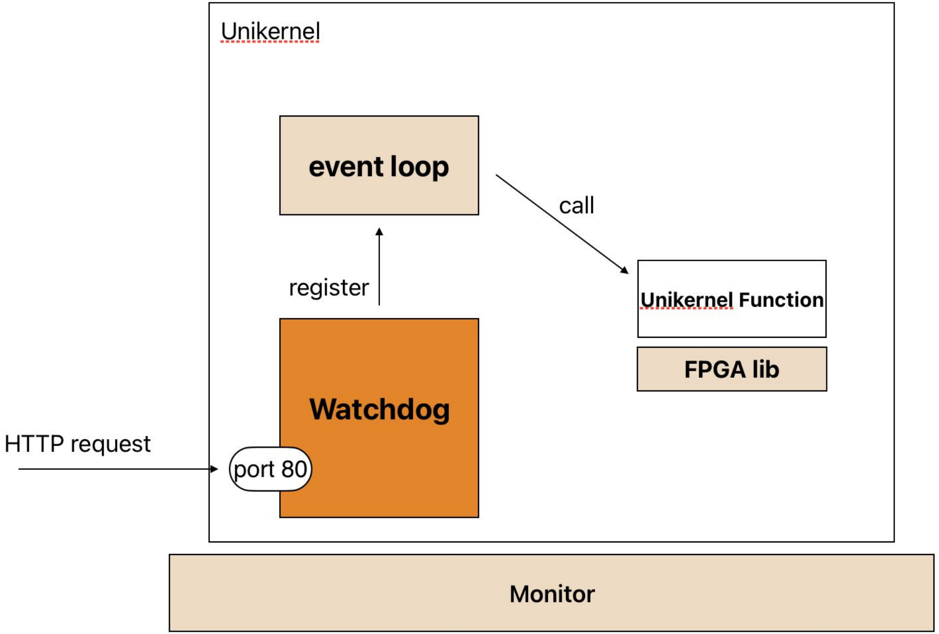
System Workflow

- Deployment
- Execution



Watchdog

- Watchdog Integration
  - event-driven execution
  - waiting for client’s request and invoke the function



Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- Evaluation
- Summary

Evaluation

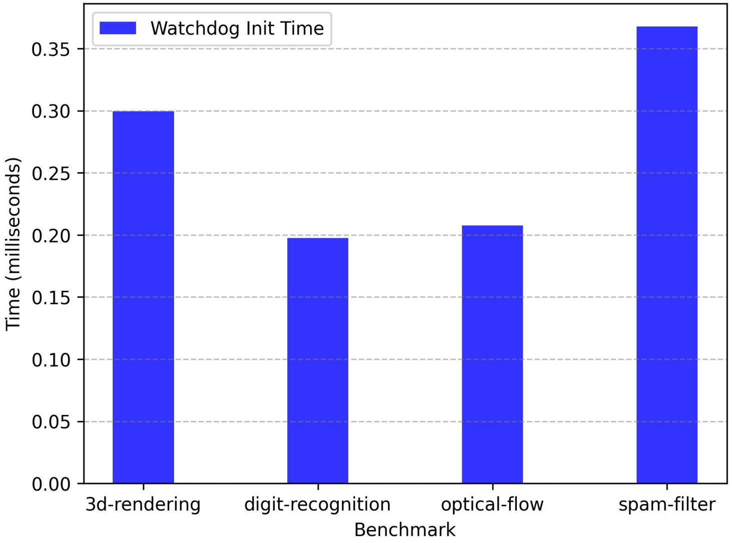
- What additional overhead does the new framework introduce?
    - Urunc deploy overhead and watchdog initiation time

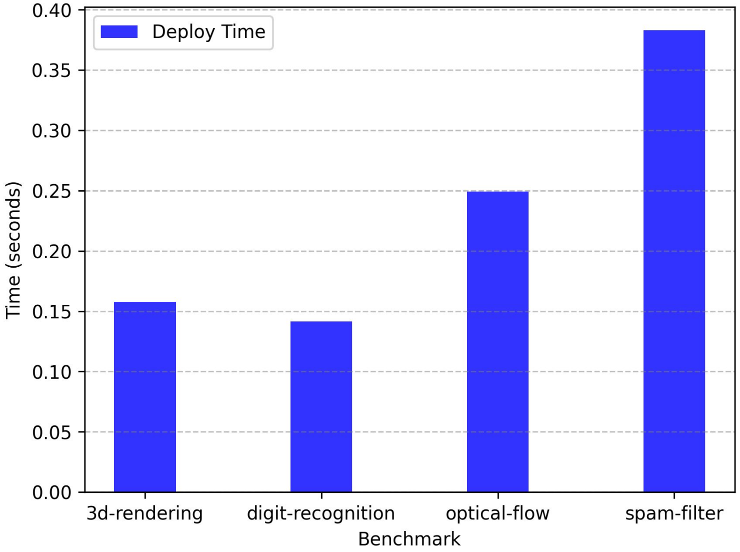
- Does the application deployed in the new framework has the same performance as original one?

Evaluation

- Experimental setup:
  - Server: Hinoki
  - Intel Xeon Gold 6238R CPU (2.20 GHz)
  - 256 GB DRAM
  - Alveo U50 via a PCIe Gen3 x16 bus.

Deployment overhead

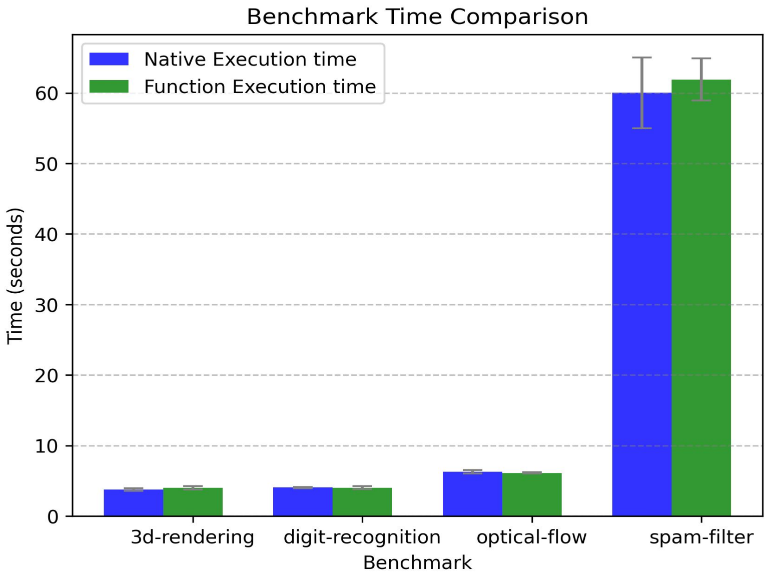




Watchdog init time is negligible compared to urunc deploy time

Performance overhead

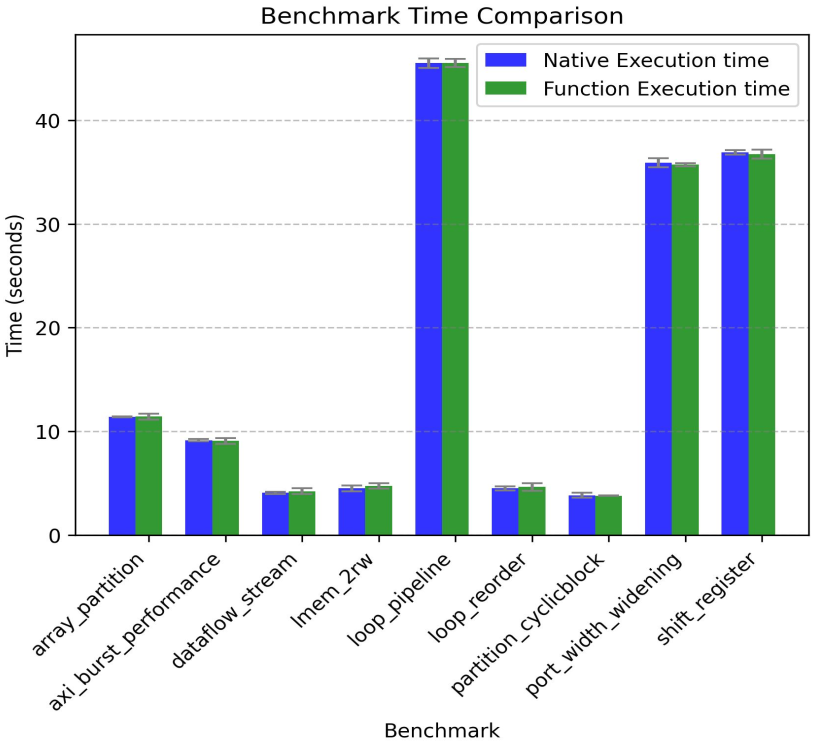
- The performance differences of Rosetta is 1.89%



Application deployed in Faas platform achieves almost the same performance

Performance overhead

- The performance differences of Vitis Accel Examples is 1.77%



Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- ~~Evaluation~~
- Summary

Summary

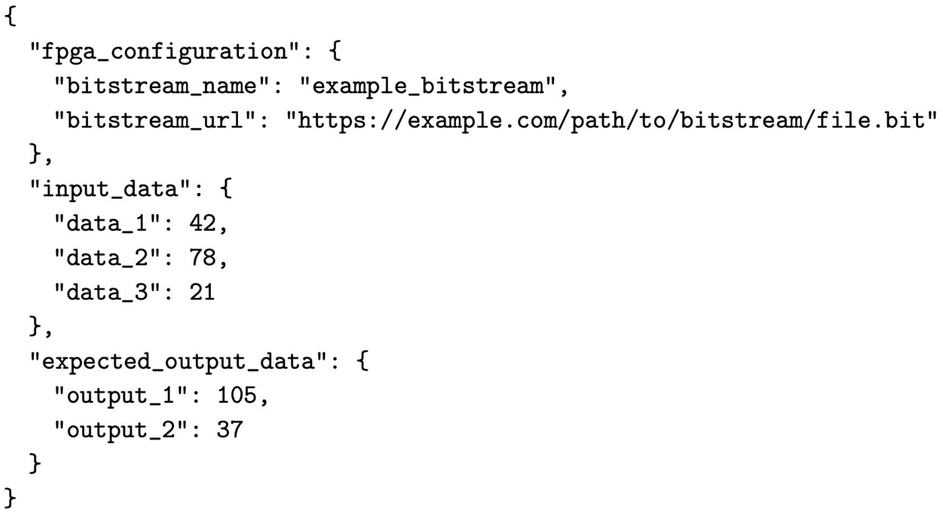
- Utilise existing resources to build a FaaS framework that support FPGAs
- Integrated Watchdog into Funky Unikernel Applications for event-driven function
- The overhead of integrating Watchdog is negligible compared to urunc deploy time
- The Application deployed in the new framework has almost the same performance as original funky application

Thank you!

Backup

FPGA Execution Request Function

- Json file as function parameter
  - User friendly: no need to care about things such as device finding and data transforming
  - Customization available: users can use their own bitstream to program the FPGA board

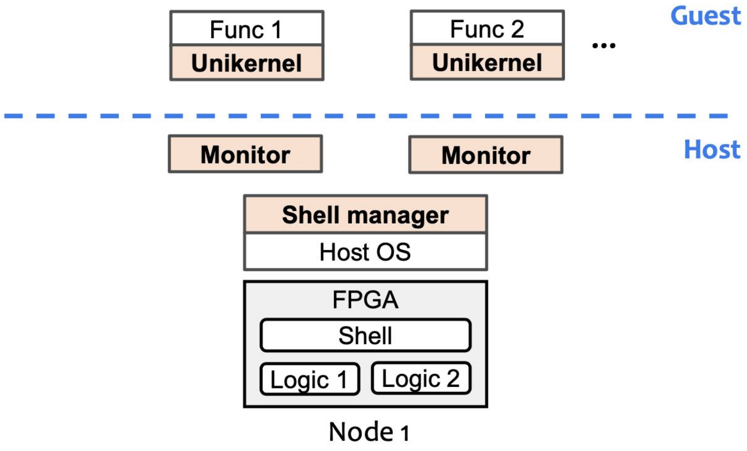


```
{
  "fpga_configuration": {
    "bitstream_name": "example_bitstream",
    "bitstream_url": "https://example.com/path/to/bitstream/file.bit"
  },
  "input_data": {
    "data_1": 42,
    "data_2": 78,
    "data_3": 21
  },
  "expected_output_data": {
    "output_1": 105,
    "output_2": 37
  }
}
```

Listing 4.1: bitstream_config.json file

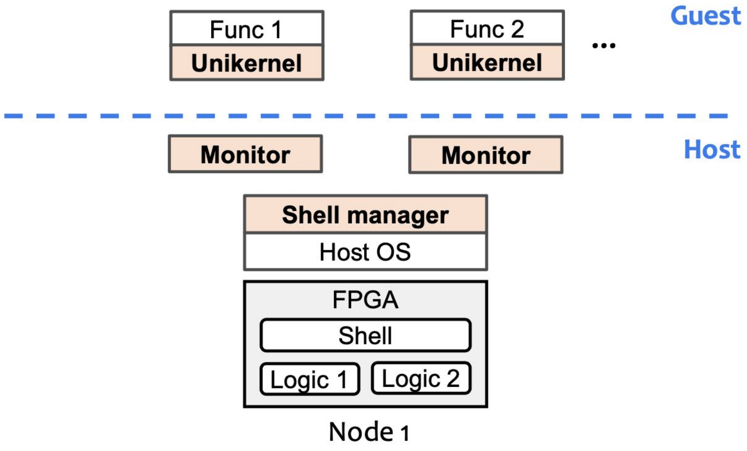
Background

- What we have
    - Funky monitor



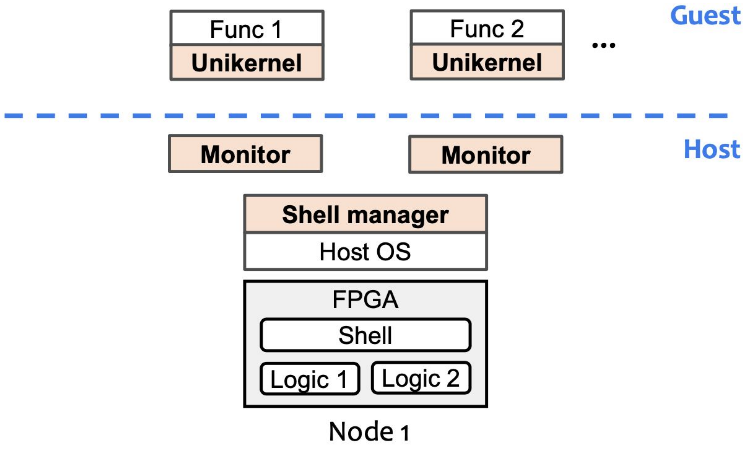
Background

- What we have
  - Funky monitor
  - vAccel Urunc

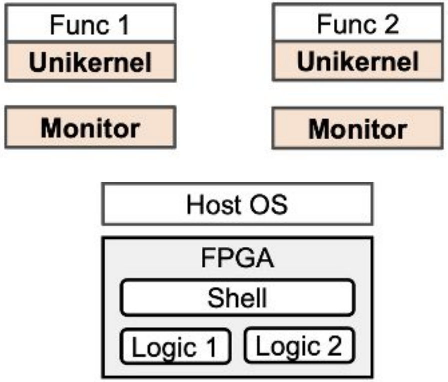


Background

- What we have
  - Funky monitor
  - vAccel Urunc
  - Kubernetes

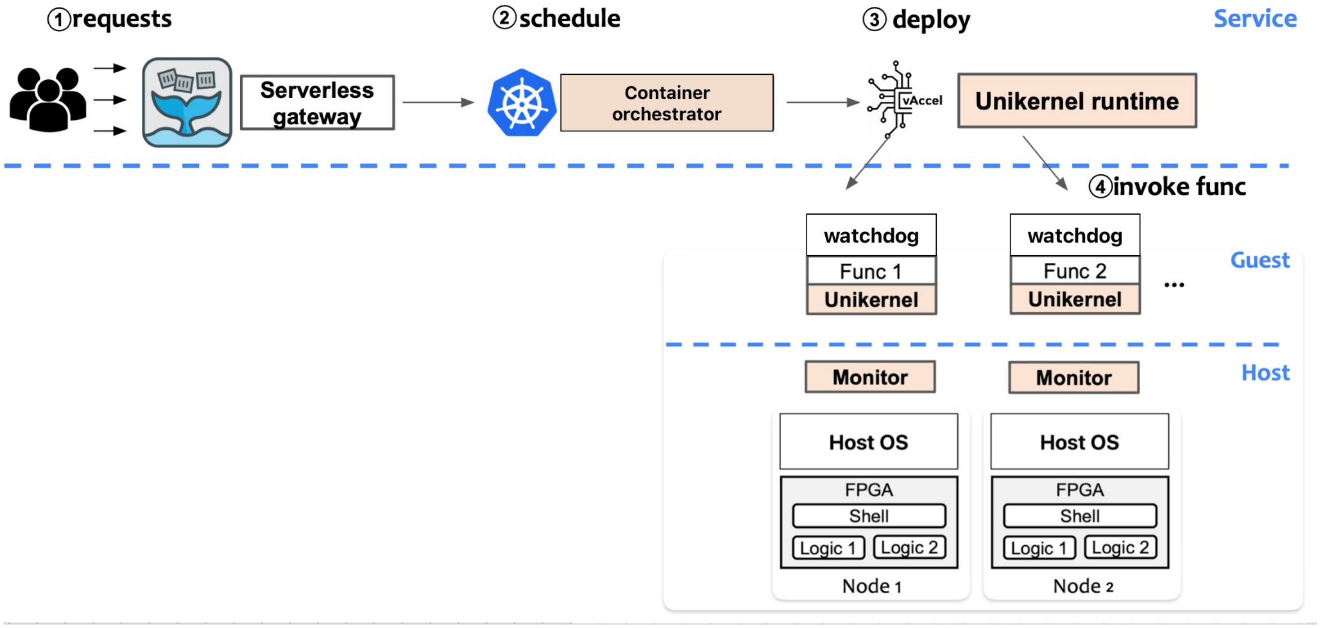


Background


Func 1
Unikernel
Func 2
Unikernel
Monitor
Monitor
Host OS
FPGA
Shell
Logic 1
Logic 2

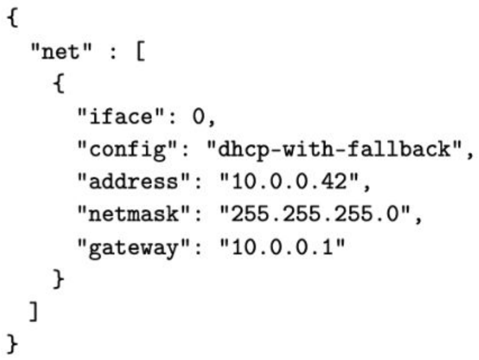
System overview

- Integrated into kubernetes world and OpenFaaS serverless

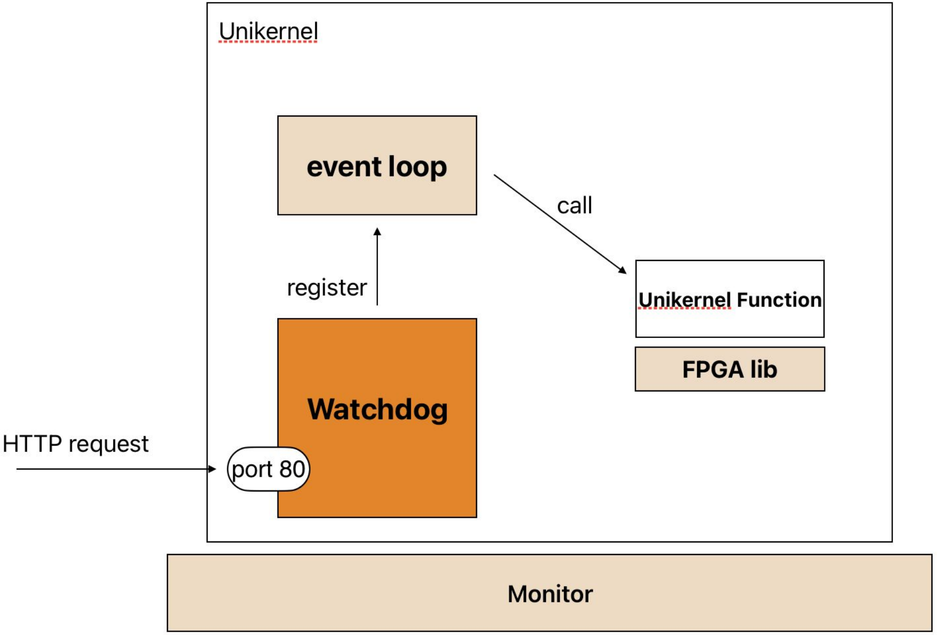


Watchdog

- Watchdog Integration
    - event-driven execution
    - configure via json file



```
{
  "net" : [
    {
      "iface": 0,
      "config": "dhcp-with-fallback",
      "address": "10.0.0.42",
      "netmask": "255.255.255.0",
      "gateway": "10.0.0.1"
    }
  ]
}
```


Unikernel
event loop
call
register
Unikernel Function
FPGA lib
Watchdog
HTTP request
port 80
Monitor