EDGELESS: A Serverless Platform Connecting Edge and Cloud

by Roman Kolcun, Chen Chen, Richard Mortier, and many others

Consortium

Disclaimer: most of figures were stolen from our partner's presentation.

Key Requirements - Programming Model

- Provide stateful execution (as opposed to stateless)
- Support various QoS
- Integrate external resources - sources of data and data sinks
- Use workflows as main deployment unit and asynchronous invocation pattern

Key Requirements - Runtime

- Reacting fast to changes in the network, node load, infrastructure, ...
- Use resource constrained devices for computation at the edge
- Supports various specialised HW (GPU, TEE, Sensors, TMU, ...)
- Optimise system based on real-time analytics of resources and activities

Key Requirements - Security

- Enable secure execution of functions (RUST & WASM)
- Support system-wide anomaly detection
- Support trust via authentication of edge nodes in the infrastructure

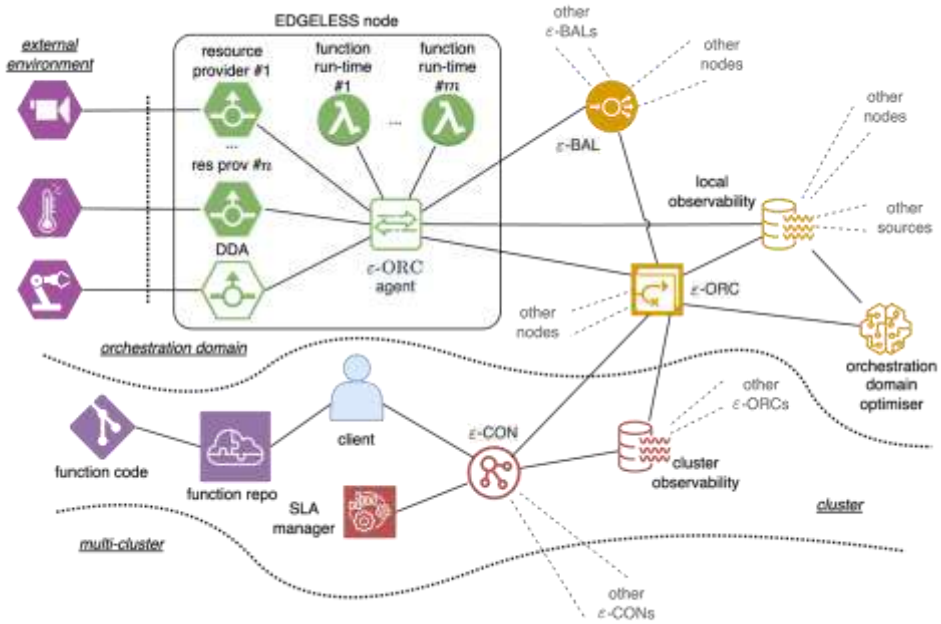
Main Concepts

- **Functions**: Basic unit of execution in the EDGELESS system
- **Workflows**: Can represent an execution of one or multiple functions in a sequence
- **Annotations**: Metadata that can be attached to functions and workflows to provide additional information about their behavior
- **Resources**: Allows workflows to interact with external environments

EDGELESS Architecture

EDGELESS cluster is managed by an ε-CON (controller) - accepts and deploys workflows into an orchestration domain

EDGELESS orchestrator ε-ORC manages the lifecycle of physical function/resource instances on EDGELESS nodes.



Functions

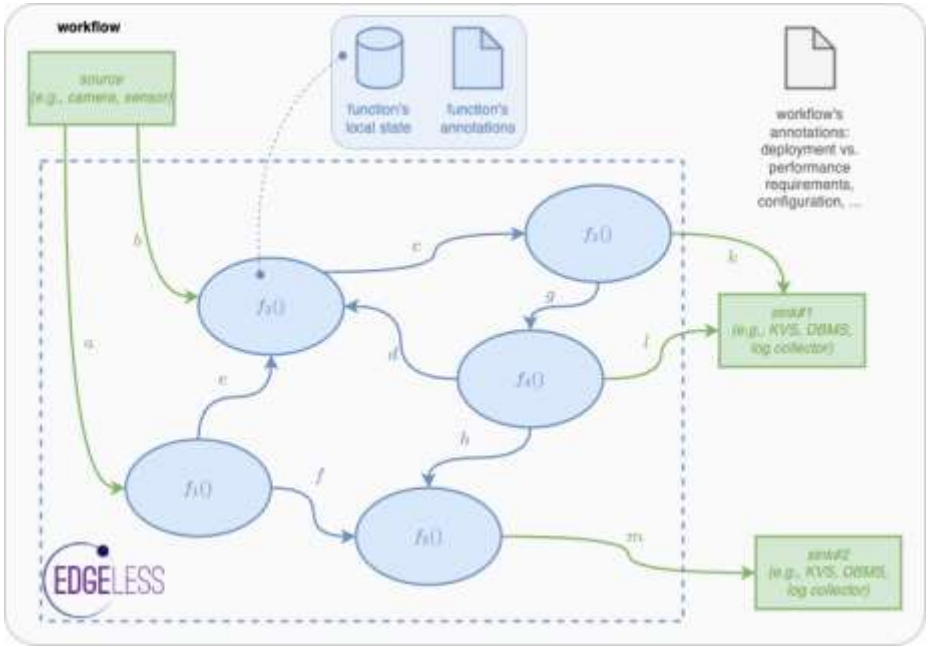
Function handlers:

- **handle_cast**: Triggered for processing asynchronous requests
- **handle_call**: Triggered for synchronous requests
- **handle_init**: Invoked when a function instance is started
- **handle_stop**: Invoked when a function instance is terminated

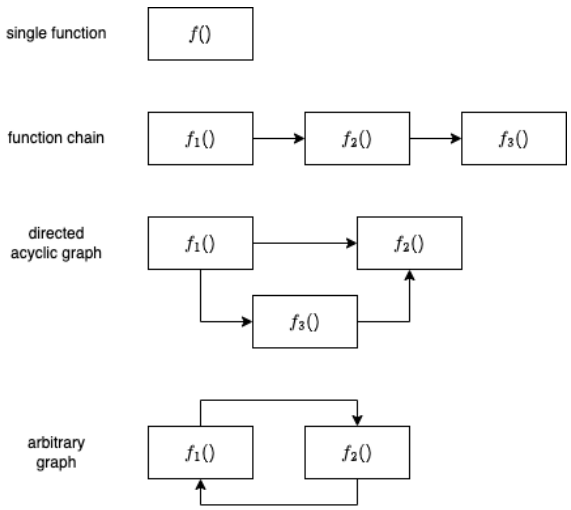
Methods:

- **cast**: send a message to a function in the workflow
- **call**: send a message synchronously to a function in the workflow
- **delayed_cast**: send a message to a function in the workflow after delay milliseconds
- **log**: produce a line of log
- **slf**: retrieve the function's own InstanceId
- **sync**: write the state to disk/database

Workflow diagram example



Workflows




```
{
    "functions": [
        {
            "name": "stage_1_function",
            "class_specification": {
                "id": "http_processor",
                "function_type": "RUST_WASM",
                "version": "0.1",
                "code": "./compiled_function.wasm",
                "outputs": ["success_cb", "failure_cb"]
            },
            "output_mapping": {
                "success_cb": "http_processor_stage_2"
            },
            "annotations": {}
        },
        {
            "name": "http_processor_stage_2",
            "class_specification": {
                "id": "http_processor2",
                "function_type": "RUST_WASM",
                "version": "0.1",
                "code": "./processing_function2/http_processor2.wasm",
                "outputs": []
            },
            "output_mapping": {},
            "annotations": {}
        }
    ],
    "resources": [
        {
            "name": "http-ingress-1-1",
            "class_type": "http-ingress",
            "output_mapping": {
                "new_request": "stage_1_function"
            },
            "configurations": {
                "host": "edgeless-project.eu",
                "methods": "POST"
            }
        }
    ],
    "annotations": {}
}
```

Annotations

- **QoS requirements**: deployment or run-time requirements enforced by the EDGELESS system, particularly the ε-CON, ε-ORC, ε-BAL, and SLA manager
- **Characteristics**: hints on the expected workflow that can be used to guide the optimisation process within EDGELESS
- **Costs**: cost-related information
- **Configurations**: operational parameters and initial settings for functions and workflows
- **Deployment Constraints**: specify the essential requirements for where and how functions and workflows are deployed

Execution Environment

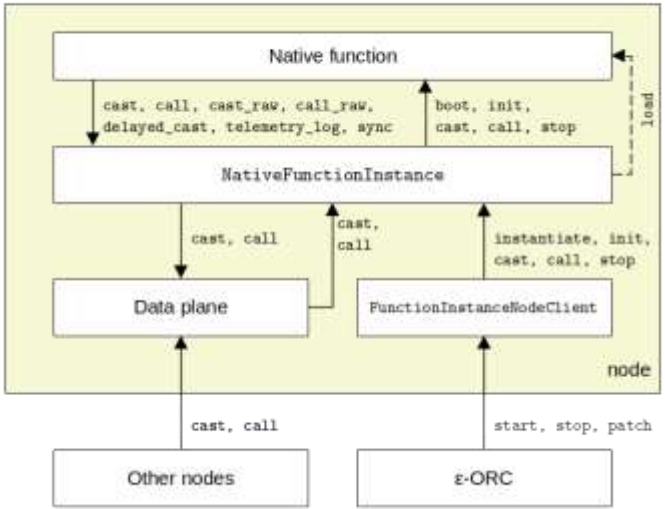
- Rust / WASM (wasmtime and wasmi runner)
- Rust / Native Execution
- Rust / Container
- Python / Container

Native Code Execution

- Allow to run EDGELESS functions directly on HW of various platforms
- Reduce start-up time
- Reduce memory requirements
- Improve execution speed by running the native code
- Virtualisation via toolchain

Native Code Execution

- Native functions are loaded using libloading crate
- Internally relies on dlopen() function
- Relatively easy to communicate between EDGELESS node and EDGELESS function
- Challenging the other way round
- Solution based on exporting pointers from EDGELESS node to EDGELESS function



Project Website

https://github.com/edgeless-project/edgeless