
KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023

WASM-Based FaaS Framework With ML Capabilities

Michael Yuan, Co-Founder, Second State

Wilson Wang, Global Edge Team Engineer, ByteDance

Introduction

Michael Yuan
Second State CEO

Wilson Wang
Global Edge Team Engineer
ByteDance

Agenda

- Cloud Application Background & Technology Selections
- Warrior FaaS Framework
- Demos

Cloud Application Background & Technology Selections

Evolution Of SW Architecture

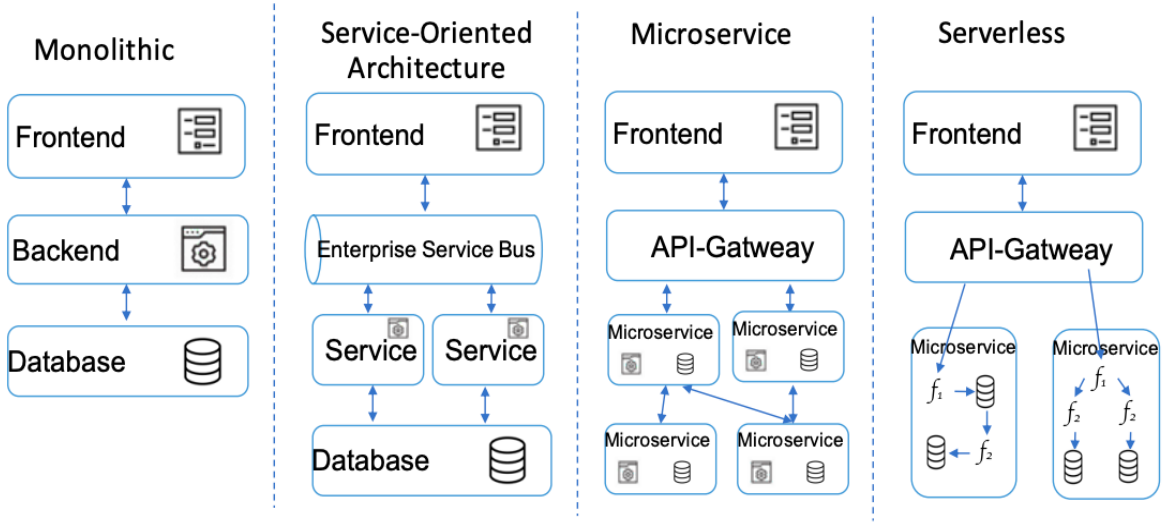
Evolution Of SW Arch

1. Monolithic
2. Service-Oriented Architecture (SOA)
3. Microservice
4. Serverless

Problems:

Smaller & divided components

- Different teams supporting different components. Hard to **collaborate.**
- Various **dependencies** block development progress.
- **Different versions** of the same application need to be aligned between teams.
- It is hard to make no mistake while working on data **Serialization/Deserialization** protocols.
- Hard to debug.

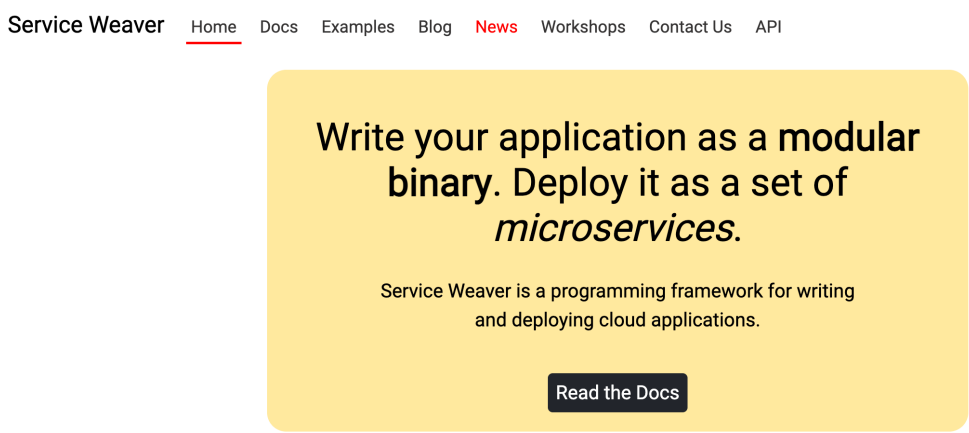


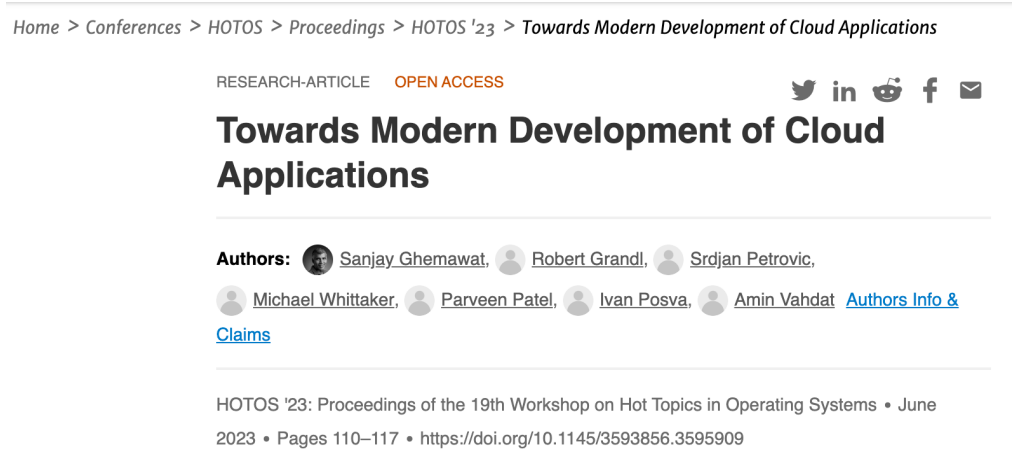
Source: D. Taibi, J. Spillner and K. Wawruch, "Serverless Computing-Where Are We Now, and Where Are We Heading?," in IEEE Software, vol. 38, no. 1, pp. 25-31, Jan.-Feb. 2021, doi: 10.1109/MS.2020.3028708.

Solutions

How to resolve these issues ?

An interesting framework：**ServiceWeaver**
And their published paper on HotOS'23





ServiceWeaver Intro

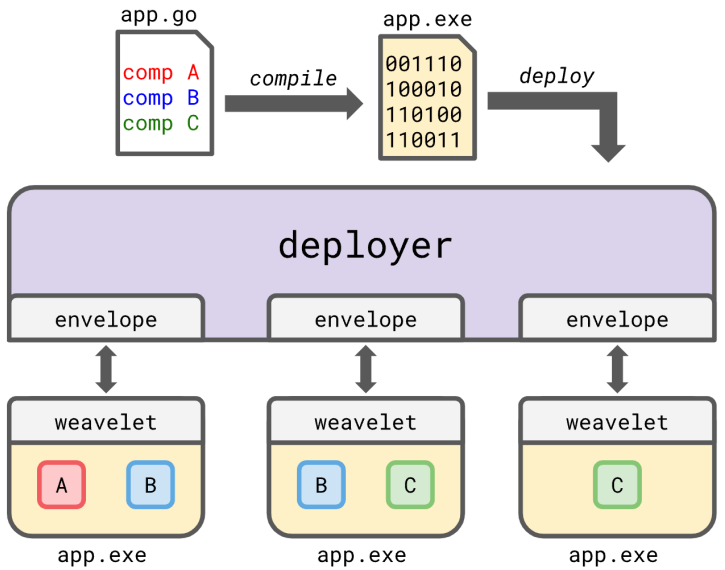
Few important points mentioned in ServiceWeaver paper

1. Develop **monolithic** applications. No interactions should happen between different versions of the same cloud application.
2. Need to identify **physical boundary** & **logical boundary** mappings. Different logical boundaries do not mean different physical boundaries.
3. A distributed smart runtime is needed for auto-optimization, auto-scaling and debugging.

Golang

Limitations of ServiceWeaver

1. **Golang only**. In cloud application development environments, interoperability between different programming languages is required.
2. Module level deployment. Not function level.



New-Gen Cloud App Development

A New Question: What will be the new generation cloud application development be like in the FaaS environment?

Our definitions of the pre-requisite:

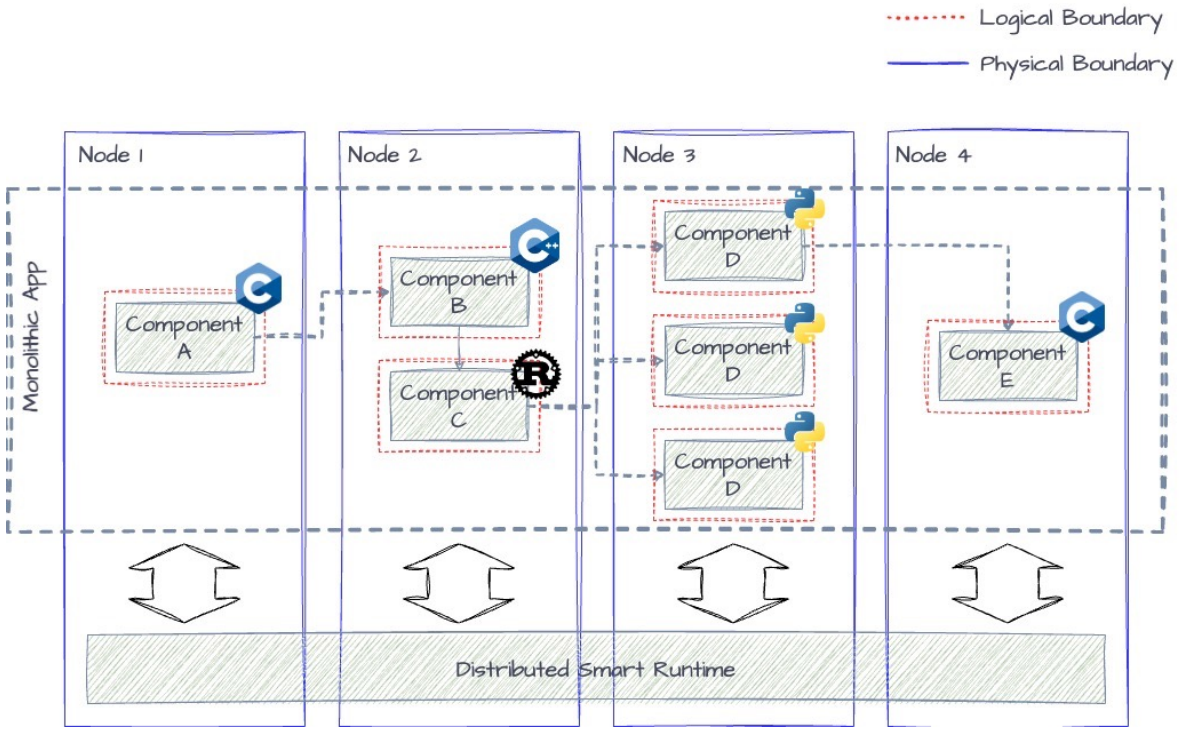
1. Similar to the concept explained in ServiceWeaver paper
2. Need to consider more enterprise scenarios.

Base on these 2 points, we bring up the goals of the next-generation cloud application development

Goals Of Next-Gen Cloud App Development

C-R-E-D-S

1. **C**ompatible**:** Works with different languages, OS, and HW platforms
2. **R**apid**:** Its start/stop time needs to be faster than major cloud application solutions.
3. **E**lastic**:** Scale up/down based on cloud application execution metrics.
4. **D**ynamic**:** Dynamically control how cloud application components are divided & deployed, as well as their # of replicas, in order to achieve goals like high resource usage or high throughput.
5. **S**ingle**: Developers need to focus on Monolithic cloud application development, not** communication details between different components, serialization/deserialization & debugging



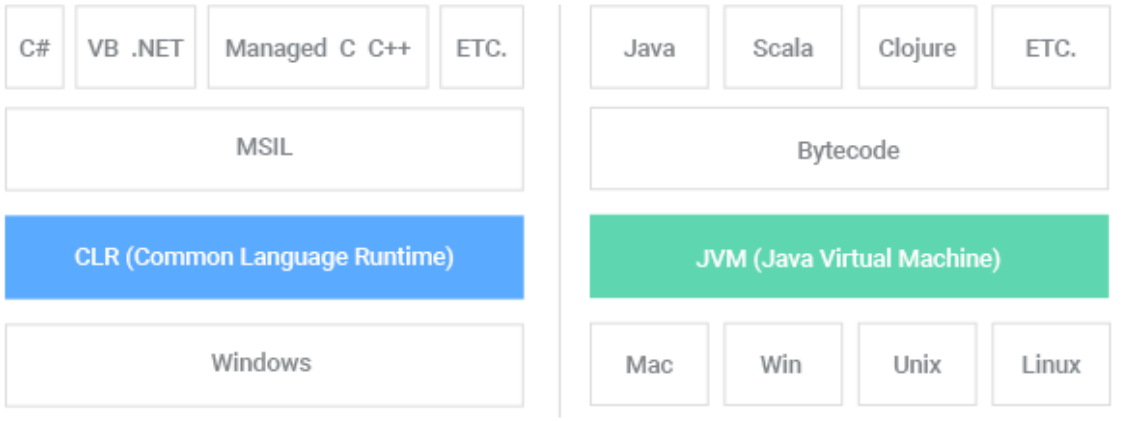
Technology Selections

How do we meet the requirement of Compatibility, Security, as well as close-to-native perf?

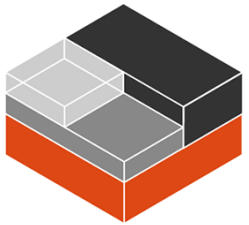
A straight-forward one is:

.NET CLR/JVM + VM/Container/Unikernel As Process

- CLR/JVM cannot make cloud application start/stop fast enough.
- For security, can we use VM/Container/Unikernel As Process? The answer is: **NO**



OverOps



LXC

QEMU

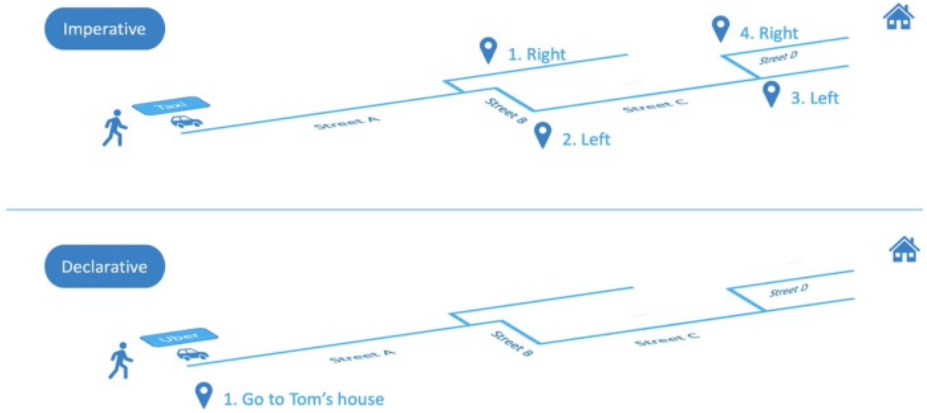
KVM

Isolation Technology Selections

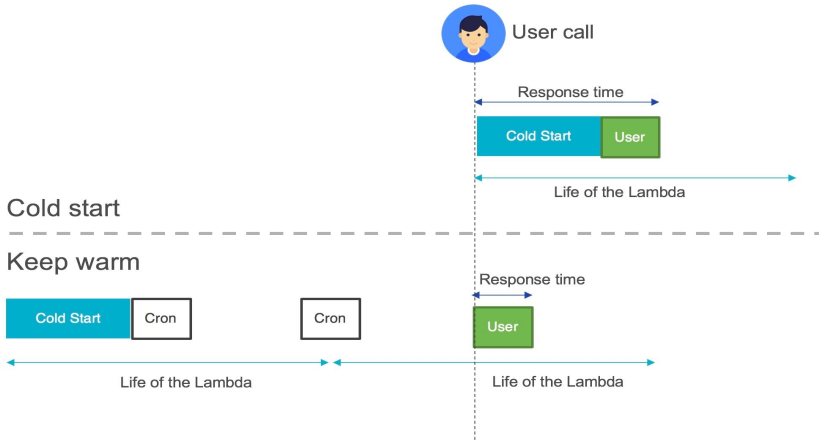
Why can’t we use VM/Container/Unikernel As Process?

Cold-Start Problem In Cloud Application

- How to **reduce cloud application cold start latency** is one of the important problems to be solved. This is related to the **isolation solution** we use.
- To solve this issue, we have 3 main directions in industrial/academic areas
  1. Modify K8s control plane from **Declarative** to **Imperative.**
  2. Use Warm Pool to speed up fetching new VM/Container instances, avoid cold start.
  3. Use Snapshot/Recovery to reduce workload loading time. Example: CRIU



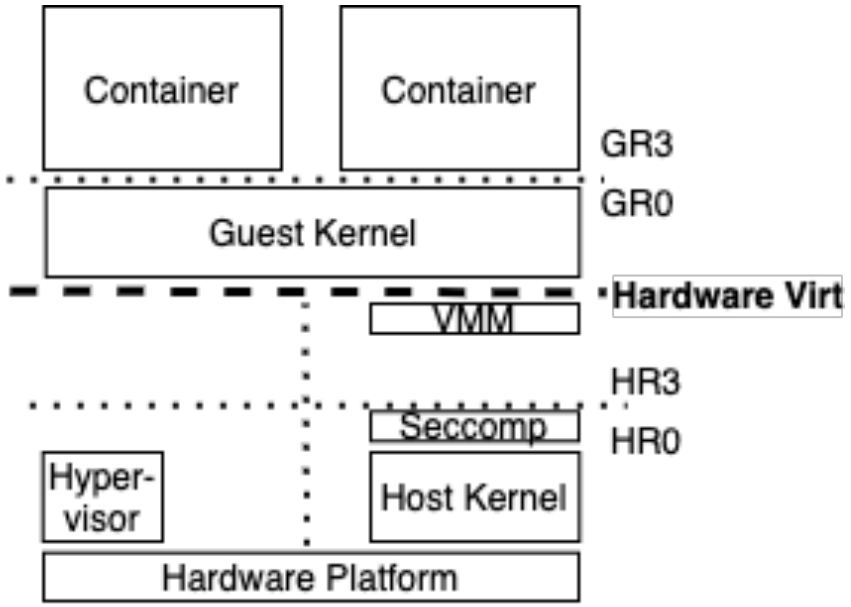
Source: https://blog.naver.com/PostView.nhn?blogId=ijoos&logNo=222144496860



Source: https://blog.octo.com/cold-start-warm-start-with-aws-lambda/

VM

- HW support: VT-x(Intel) & SVM(AMD)
- VMCS stores VM states
  - VMEnter/VMExit takes ~ 1,500 CPU cycles
- Memory Access
  - Guest User -> Guest Kernel -> Host User -> Host Kernel
  - HW support has EPT/NPT acceleration, close to native perf.
- Current cloud native solutions:
  - KubeVirt
  - Kata Containers
- Large images, high security level, slow boot up.

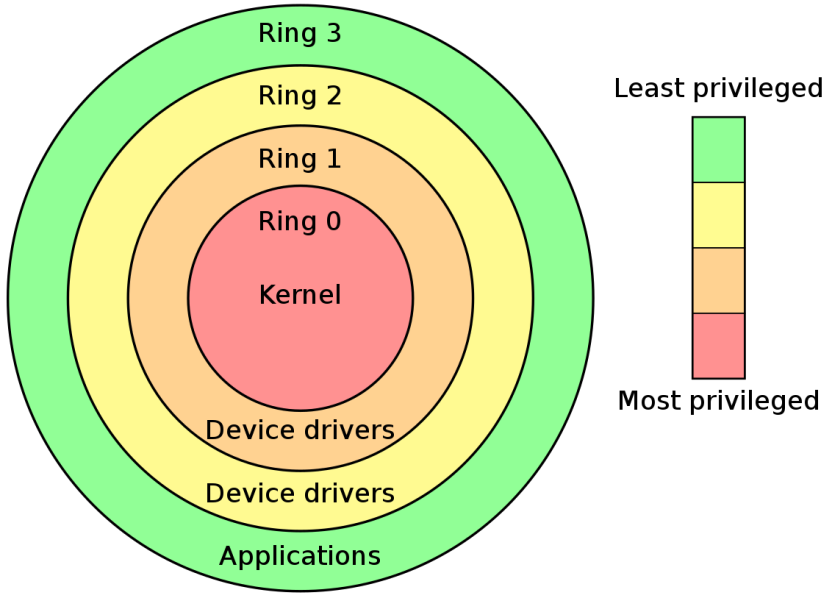


kata containers

KubeVirt

Containers

- HW support: Protection Ring (Ring 0 ~ 3)
- Ring switching triggers dumping of register values
  - A system call can take ~200 CPU cycles
- Memory access requires Page Table translation
  - Page faults are handled by OS Kernel.
- Medium image size, medium level security, faster boot up.

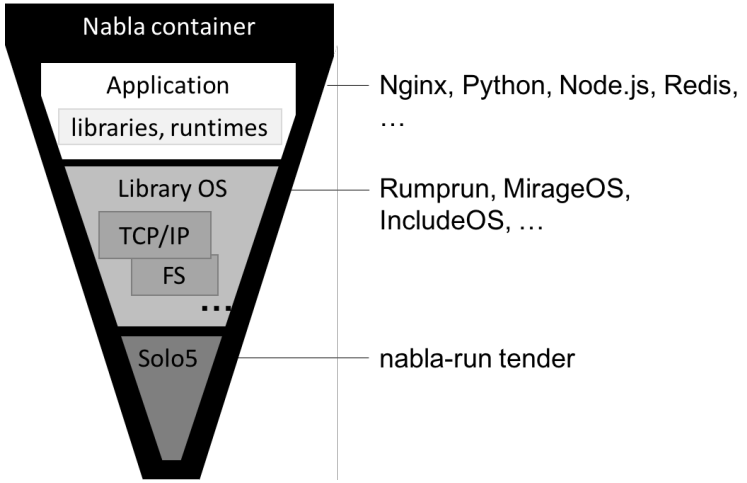


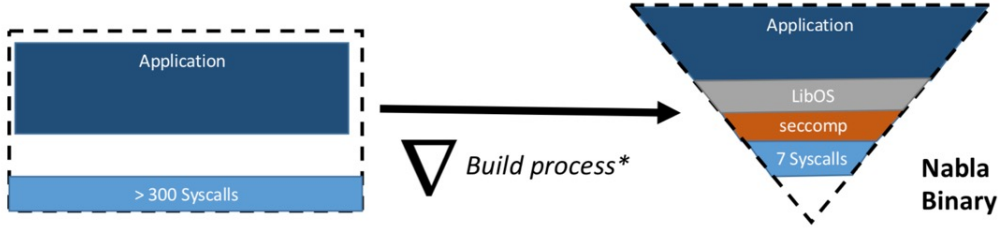
containerd

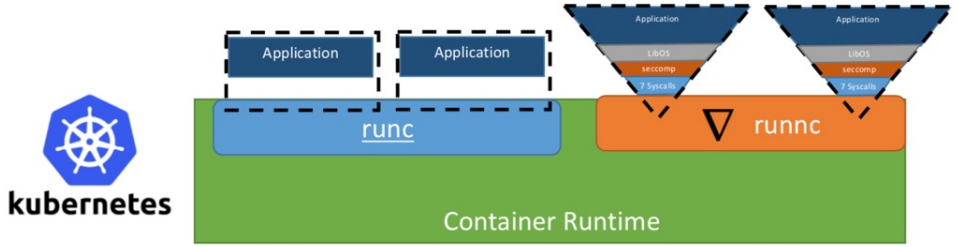
cri-o

Unikernel As Process

- Unikernel is a big category. Here we only discuss Nabla Container and its similar alternatives.
- No HW isolation feature needed
  - Reduced system call to reduce attack surface.
  - Still, there are around 7 system calls are needed.
- CALL/JMP instructions are used mostly instead of Syscall.
  - ~20 CPU cycles
- Tiny image size, security is required from external(Example: Linux seccomp), super fast boot up.



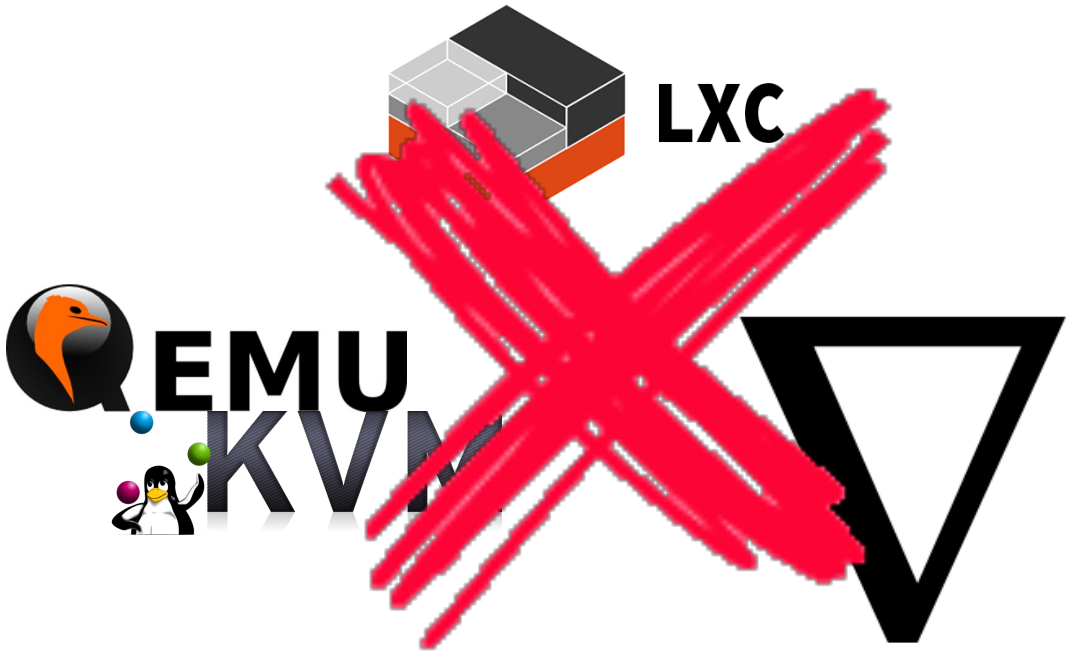




Solutions Comparison

VM, Container & Unikernel As Process cannot bring high performance especially when lightening start up time is required. No matter how we optimize cold start, we still can observe:

- VM startup > 10x Container startup
- Container startup > 10x Unikernel As Process startup



Does this mean we need to use **Unikernel As Process**? The answer is still **NO**. Because:

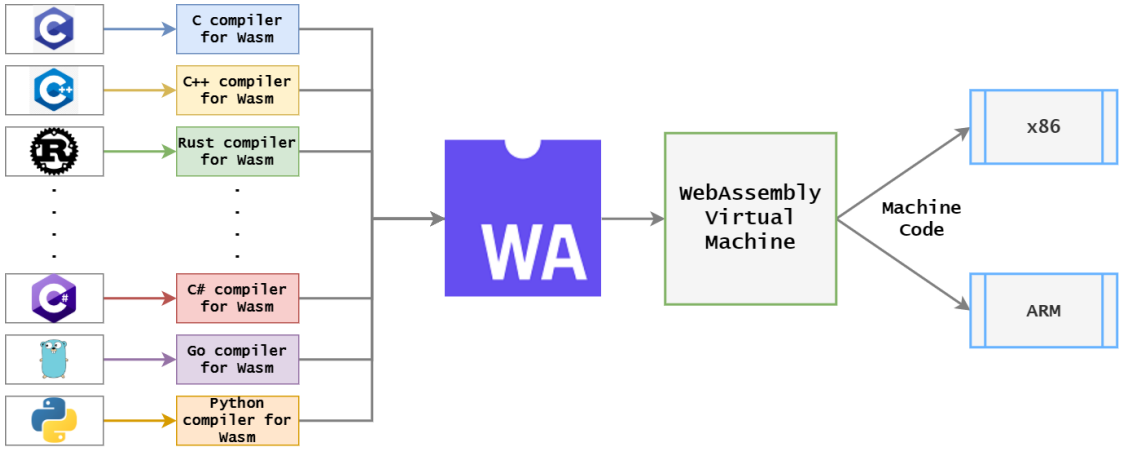
- Not compatible to different platforms/environment
- Limited **parallel execution** support (thread/process)
- Hard to debug

Final Selection

The final isolation solution selection is:

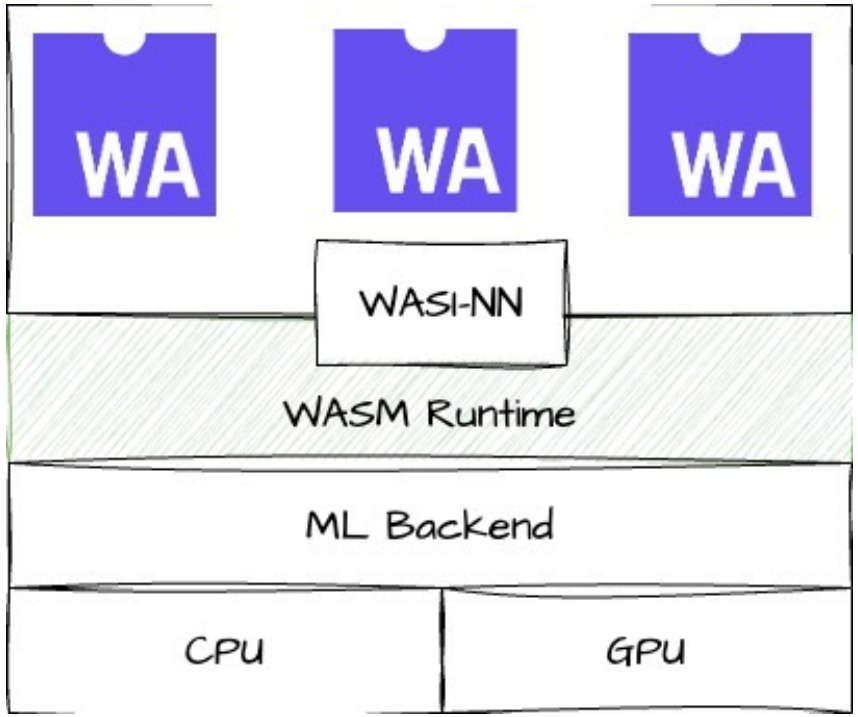
WebAssembly

- In **security** area, It uses **SFI**(Software Fault Isolation)
- Low overhead when compared with Python & JS
- **Easy to migrate to different platforms/OSes, HW agnostic**
- Easier to debug



AI & WASM

- In WASM, There are multiple way to run **ML inference**.



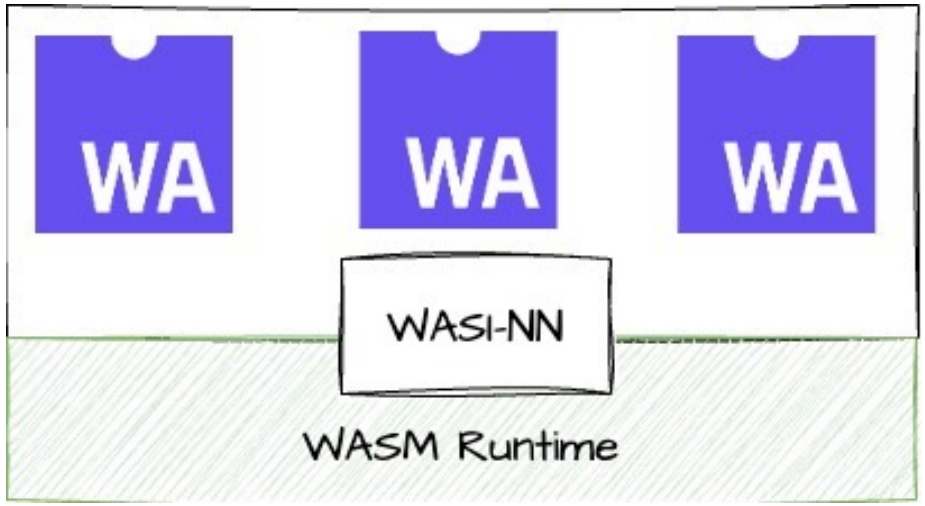
Host

- **Use WASI-NN, HW/OSes/ISAs are transparent to user code**
- Available HW accelerations: CPU, GPU, TPUs, etc.
- Base on SW/HW, support different backends to improve ML performance.

AI & WASM

WASI-NN

- Started in 2019 as an official WASI extension
- Enables use of host functions to perform AI inference
- Currently in phase 2
- Implemented by Wasmtime, WAMR & WasmEdge



https://github.com/WebAssembly/wasi-nn

AI & WASM

WASI-NN support in WasmEdge

WasmEdgeRuntime

- Fully standard compliant
  - Support emerging features
  - Contributed the current: Rust API crate
- **Multiple backends** available (plugins)
- Data pre-/post-processing SDKs
- Plugins are **Component-Model ready**
- JavaScript & Python APIs are in the works

https://github.com/WasmEdge/WasmEdge

AI & WASM

WASI-NN Ecosystem

Support depends on the specific runtime.

- Backends
  - OpenVINO
  - TensorFlow
  - Pytorch
  - Ggml

- Data processing
  - OpenCV
  - FFMpeg

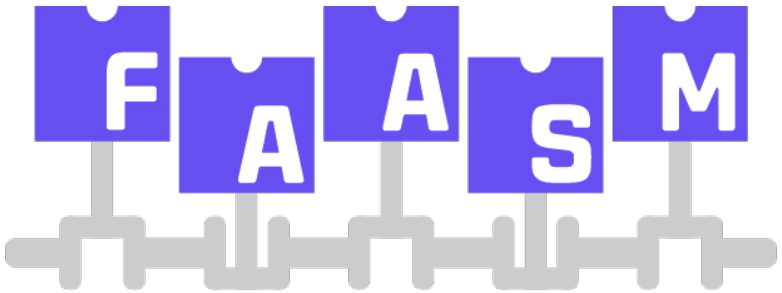
- Models
  - Llama
  - Yolov5
  - MediaPipe

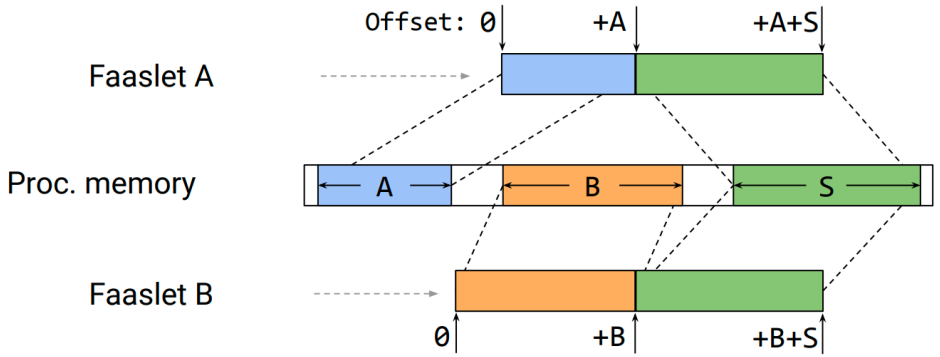
Distributed Computing Engine Selection

Wait a sec, did we miss anything?

We didn't even mention how to achieve **E**lastic, **D**ynamic, and **S**ingle in C-R-E-D-S.

- Solutions such as K8s cannot meet our requirement of **monolithic** application development.
- FAASM's paper is interesting. It meets most of our needs(Monolithic, Distributed Scheduling, Distribute Object Storage). However, it is an academic area production. It is Task-Oriented, not Data-Oriented hence not a good solution for ML workloads.





Technology Selection

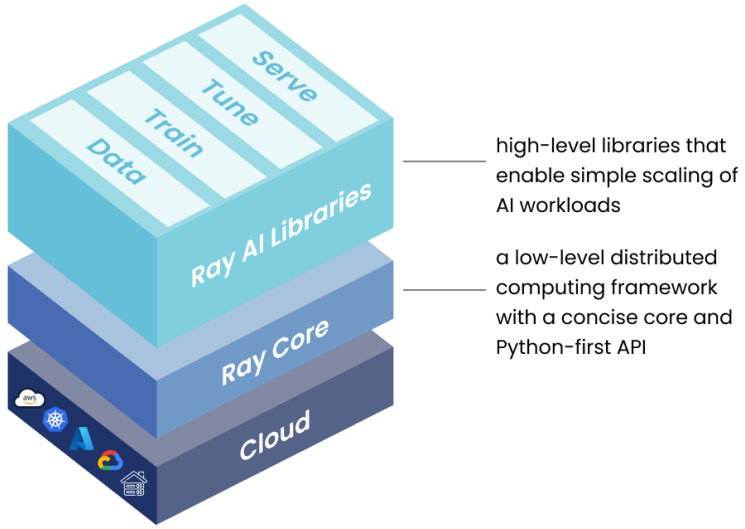
Finally, we chose Ray framework as our distributed computing engine.

RAY

What is Ray?

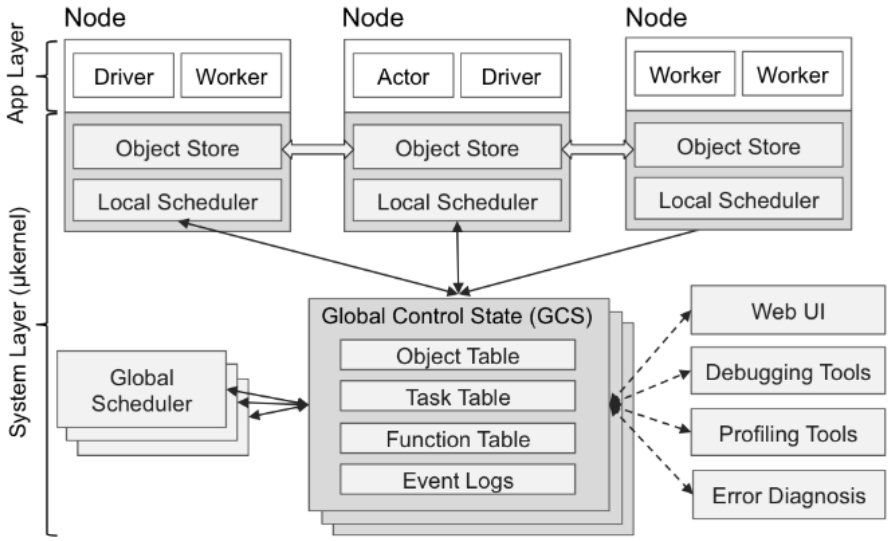
Ray is an open-source unified compute framework that makes it easy to scale AI and Python workloads — from reinforcement learning to deep learning to tuning, and model serving.

According to Ion Stoica Ray is a "**distributed computing ecosystem as a service**"



Ray Framework

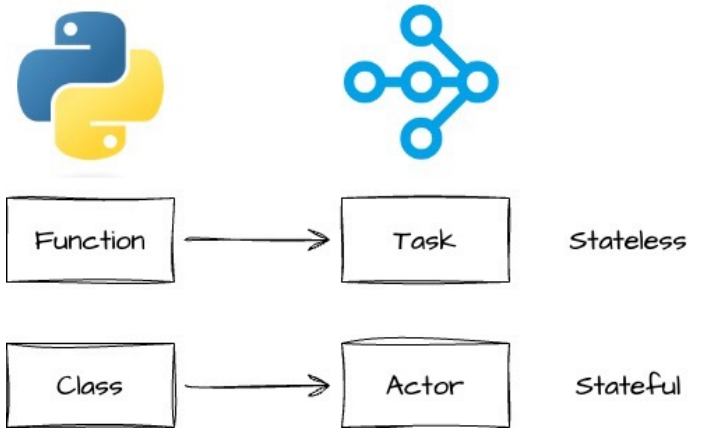
- Ray cluster roles
  - one head node + several worker node
- Raylet acts like K8s Kubelet.
- GCS acts like K8s APIServer.
- Object store holds all distributed objects.
- Ray Example

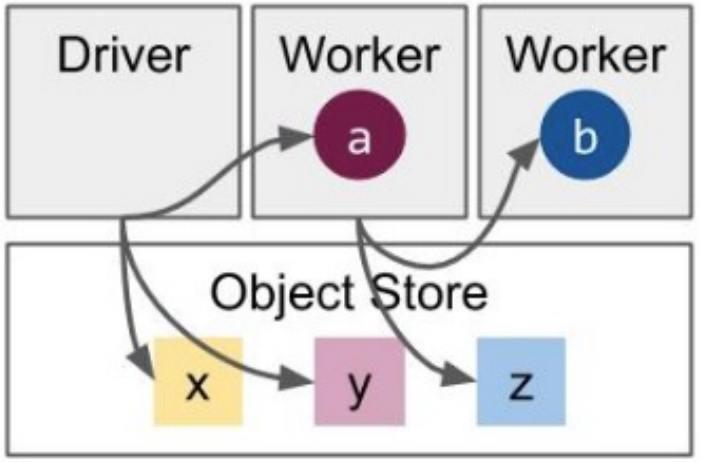



```python
# Define the square task.
@ray.remote
def square(x):
    return x * x

# Launch four parallel square tasks.
futures = [square.remote(i) for i in range(4)]

# Retrieve results.
print(ray.get(futures))
# -> [0, 1, 4, 9]
```

Needed Features From Ray

- **Remote Invocation**: Ray accepts job invocation requests and schedule jobs on proper nodes.
- **Object store**: Jobs can save variables objects into object store and job dependencies can be built upon the states of these objects (ready or not). These dependencies can help Ray decide job scheduling.
- **Actor model**: Jobs can be stateful(Actors) or stateless(Tasks). This is convenient abstraction for developers.
- **Monitor executions metrics** for better online/offline scheduling.





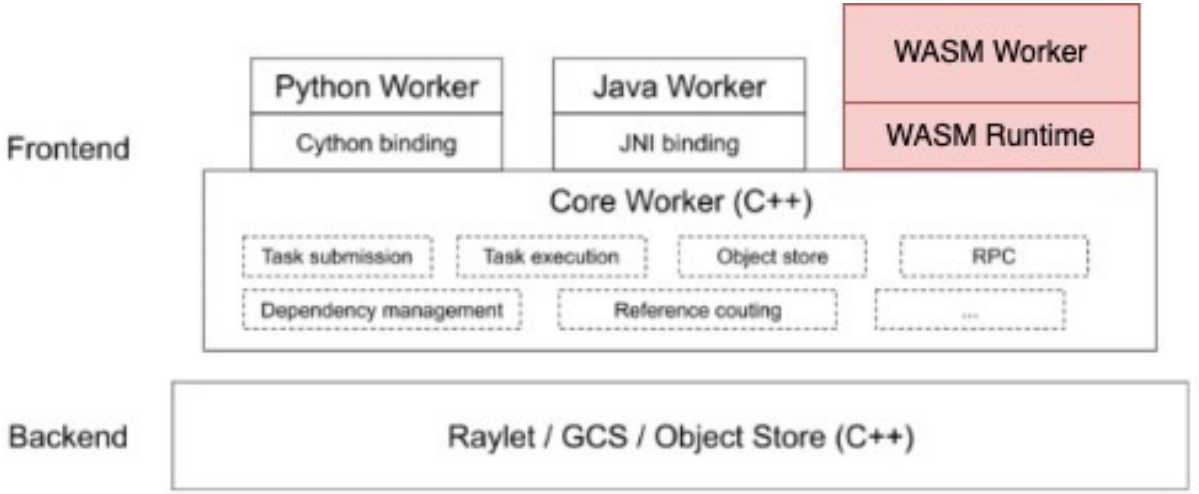
Warrior: A FaaS Framework Based-On WASM

Modifications To Ray

Warrior provides Ray distributed computing capabilities through WASM hostcalls

Implementation consists 3 major parts

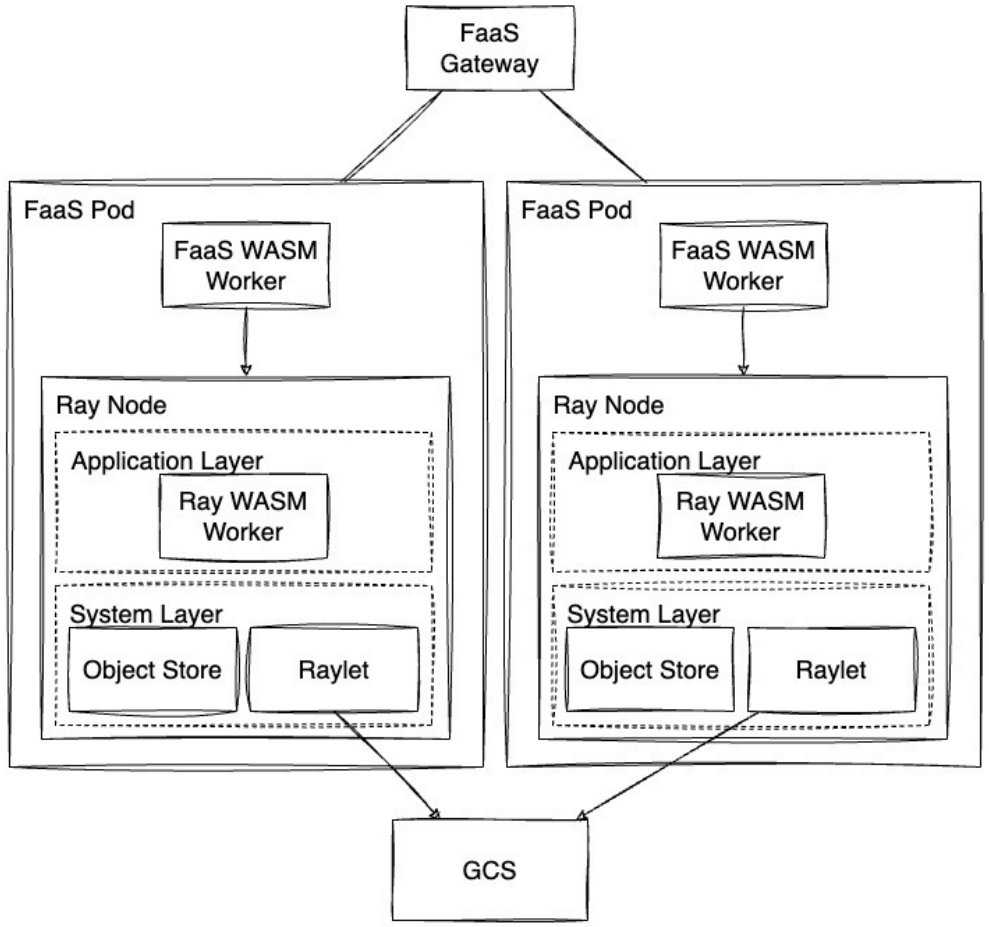
- **Ray WASM worker:** Execute WASM jobs on Ray framework, similar to Ray's Java, Python, C++ workers. Based on WasmEdge
- **WASM command line tools**: Establish connections to Ray. Load and execute WASM modules
- Integrated with WasmEdge plugin systems. Support multiple extensions including WASI-NN



WasmEdgeRuntime

Modifications To Existing FaaS Architecture

- **Integrated with WasmEdge, Ray framework & Volcano Engine FaaS platform. Most of the FaaS platform can support Warrior after some modifications.**
  1. Original FaaS requests need to be modified to support WASM.
  2. Run Ray clusters somewhere near the FaaS cluster, to enable Ray support.
- The reasons for choosing **WasmEdge** as our WASM runtime:
  - Easy to integrate with other **non-Rust** projects.
  - First tier performance among different WASM runtimes.
  - Plugin support enables ML inference workload execution.



†: Volcano Engine is not required, can be integrated into other FaaS Systems.

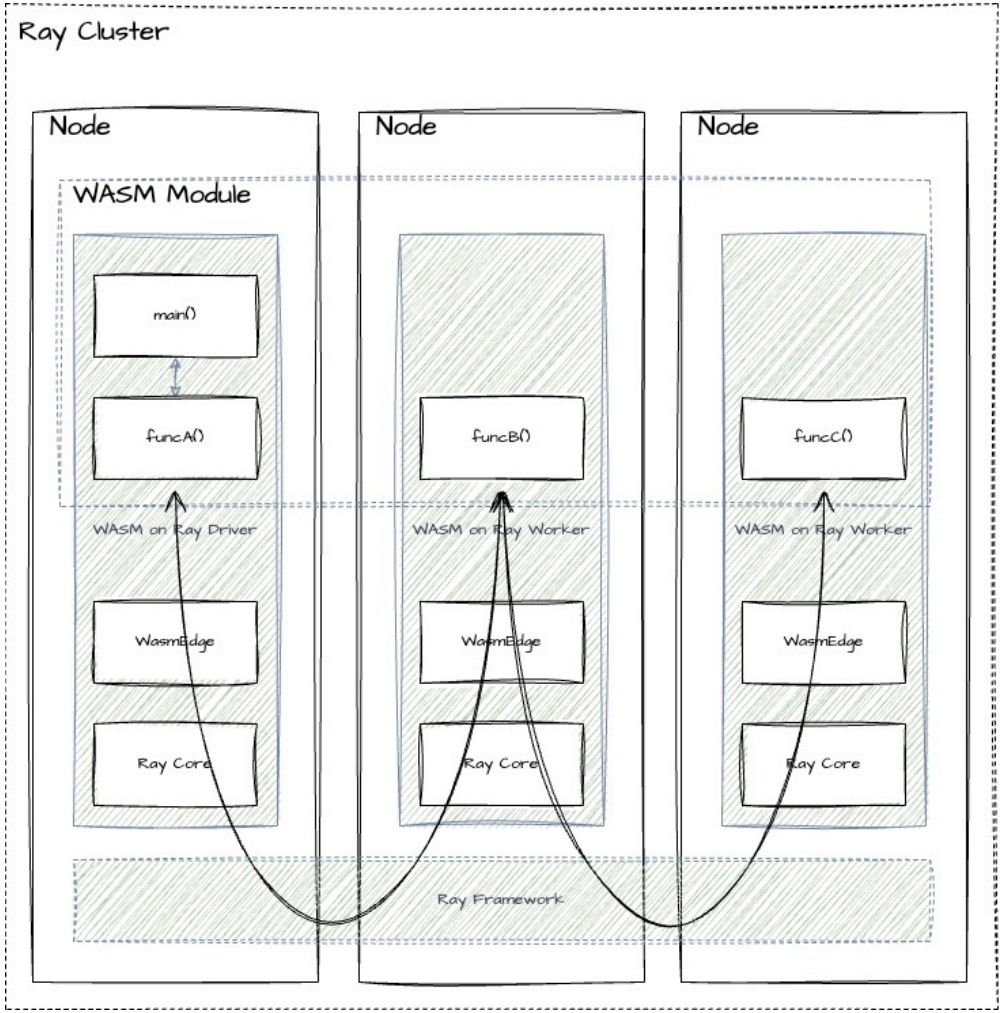
VolcanoEngine

WasmEdgeRuntime

Invocation Example

Distributed Invocation: WASM application uses Ray to invoke a task/actor. Task/Actor invocation can be on the same/different Ray node, depending on performance metrics.

1. **Loading Application**: An incoming request initializes a FaaS application on one of the FaaS worker.
2. **Local Invocation**: Application calls one of its local function funcA() during execution.
3. **Remote Invocation**: In funcA(), it asks Ray to invoke funcB() through WASM hostcall. Ray cluster finds a proper worker node to execute funcB().
4. **Nested Remote Invocation**: funcB() can invoke another function funcC() through WASM provided Ray hostcall.
5. **Result Return**: Upon completion, funcC() returns to funcB(). After funcB() finished, funcA() can process funcB()'s return result.



Example Code

Local call (first half of the file)

```c
#include "assert.h"
#include "stdio.h"
#include "war.h"

// export buffer free() to host
ENABLE_WAR_BUFFER_AUTO_FREE

// export buffer alloc() to host
ENABLE_MALLOC_BY_HOST

float task_func(char a, short b, float c, float d, int e) {
  float res = a + b + c + d;
  return res;
}

int main() {
  float result;

  printf("call task_func directly\n");
  result = task_func('a', 1, 2.0, 3.0, 4);
  printf("result: %f\n", result);

```

Normal Func Call

VS

Remote Invocation (Second half of the file)

```c
  int res;
  size_t res_len = sizeof(float);
  warbuffer *wbuf;
  // initialize ray engine
  WAR_INIT();

  // allocate buffer to store remote invoca    ref handle
  WAR_OBJID_ALLOC(wbuf);

  printf("call task_func remotely\n");
  // invoke function remotely
  res = rcall(wbuf, task_func, 'a', 1, 2.0, 3.0, 4);
  assert(res == 0);
  // get function result from ref handle
  res = rget(wbuf, &result, &res_len);
  assert(res == 0);
  printf("result: %f\n", result);

  WAR_OBJID_FREE(wbuf);
  return 0;
}
```

Distributed Func Invocation

- Immediate call & return.

- Hostcall returns a reference to the invocation.
- Wait for the result and copy result back

For more details, visit source: https://github.com/wilsonwang371/ray/blob/wilson/wasm-on-ray/wasm/example/simple/simple.c

Demo

- Simple function invocation
- MobileNet inference

Future Work

- Design and implement more transparent remote function invocation
- Execute Llama2.c on Warrior platform.
- SDKs for different languages.

Thanks!