Envoy/Istio WebAssembly 扩展实现原理

边鹏远 (github: bianpengyuan)

目录

- Wasm 和 V8 engine 简介
- Wasm 扩展对于Envoy的意义
- Wasm 扩展在 Envoy 的实现原理
- Wasm 扩展在 Istio 的应用
- 未来工作

WebAssembly

“WebAssembly (abbreviated Wasm) is *a binary instruction format* for *a stack-based virtual machine*. Wasm is designed as a portable *compilation target for programming languages*, enabling deployment *on the web* for client and *server applications*.”

- 二进制指令集
- 操作堆栈虚拟机
- 多语言编译对象
- W3C 标准，所有主流浏览器都支持
- 代替 asm.js 和 NaCl
- 不仅面向web，可以在任何地方执行

WebAssembly 用途

- 高性能需求
  - 编解码，游戏，多媒体编辑
- 利用现有的C/C++代码库
- 使用任意语言开发Web应用, Rust, AssemblyScript 等等
- 嵌入其他应用: Node.js, CDN 等等

WebAssembly 二进制格式

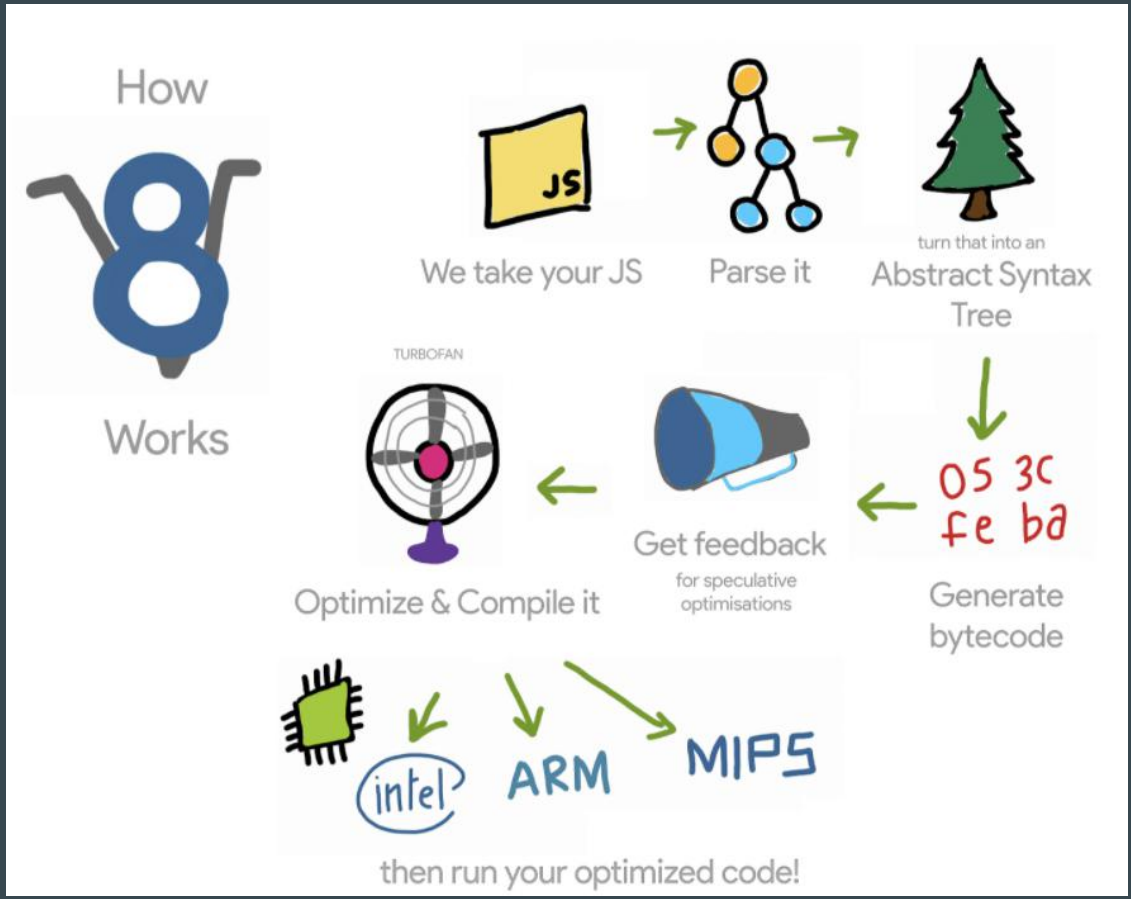
- Wasm 二进制文件 -- Wasm module
- 一个 Wasm module 包括
  - 函数
  - 全局变量
  - 引入和输出的函数
  - 内存 (linear memory) 要求

WebAssembly 简洁的指令集

- 只有四种数据类型
  - i32, i64, f32, f64
- 指令都是基于栈的操作
  - 例如 i32.add 从栈取两个值相加然后压入栈中
- 从内存读取数据
  - i32.const 1024
  - i32.load
  - 从线性内存位置 ‘1024’ 读取i32数据

V8

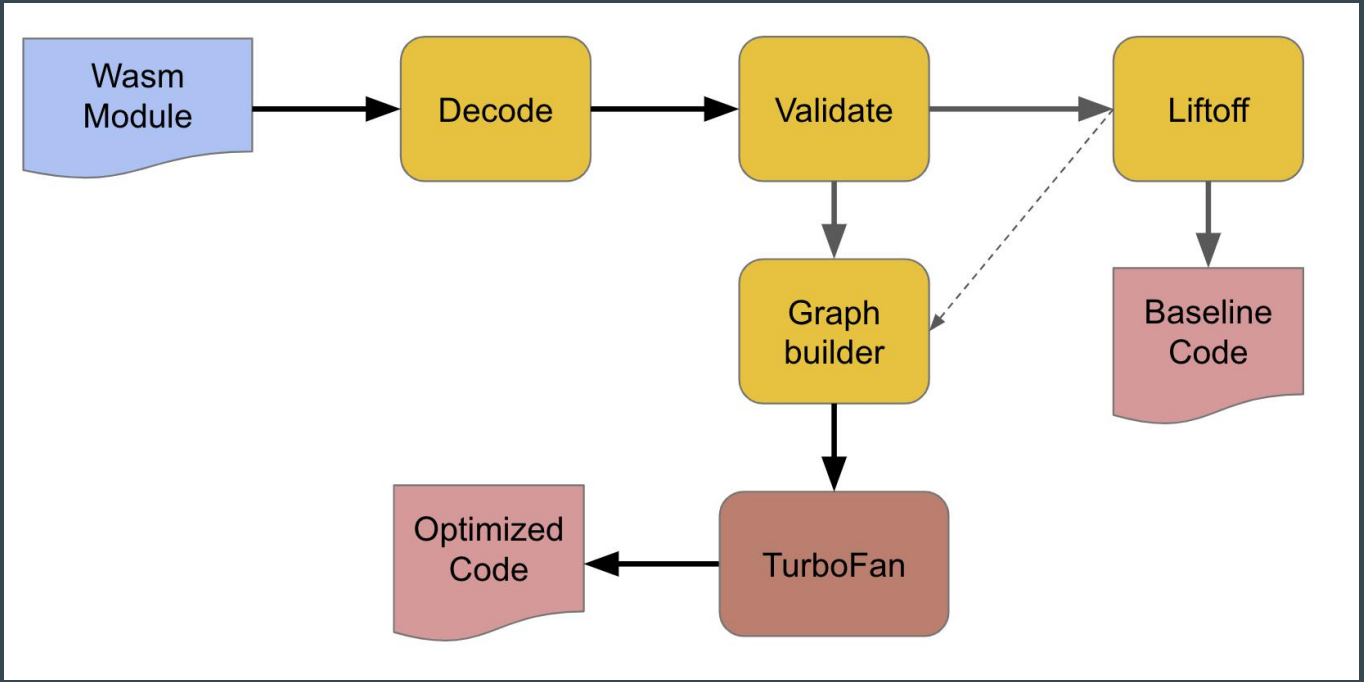
- JavaScript 解析，编译，运行，Debug，性能分析
- 内存管理 (GC)
- 内置JavaScript library



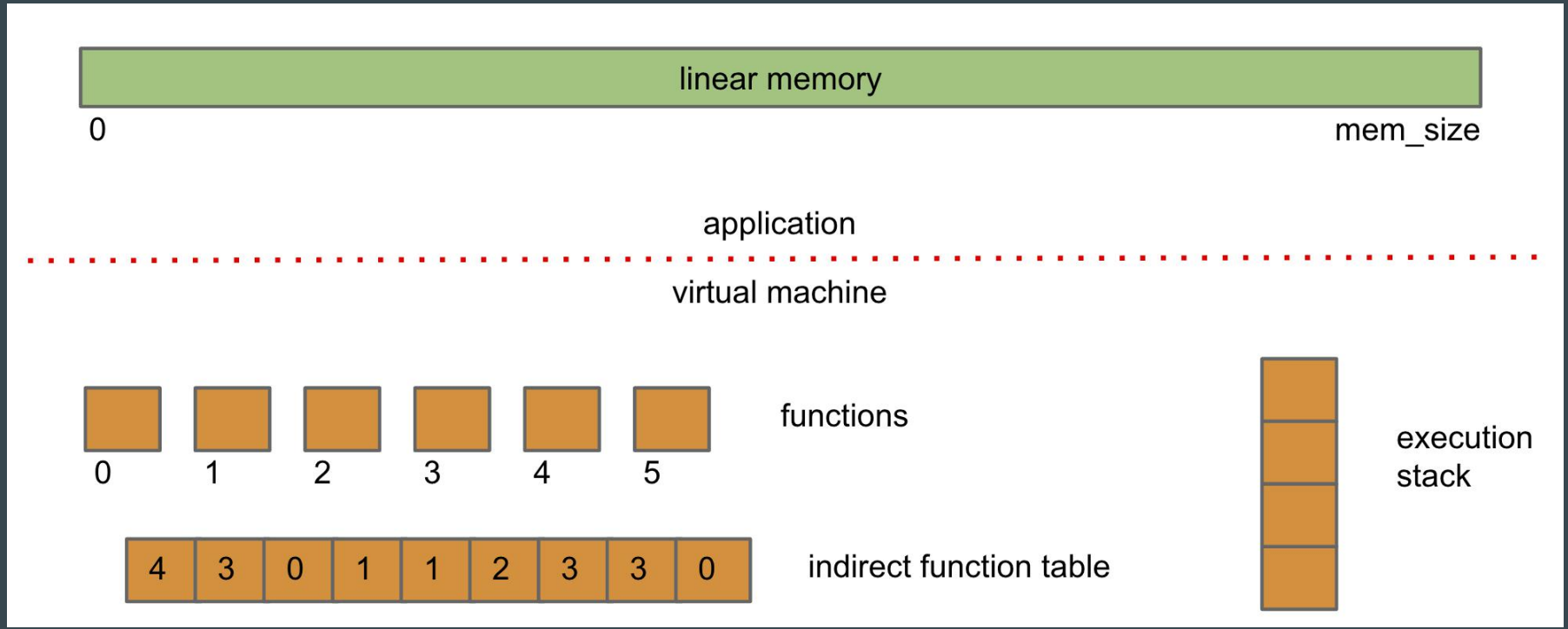
V8 Isolate

- 一个VM实例，Isolate 之间相互隔离
- 拥有自己的heap
- 只能由一个线程执行
- 在 Chrome 里面，一个 Isolate 对应一个 tab

V8 如何运行 Wasm module



V8 如何运行 Wasm module



Wasm 对于 Envoy 的意义

Envoy具有很强的可扩展性，可扩展地方包括

- L4 / L7 filters
- 访问日志，链路追踪，监控数据

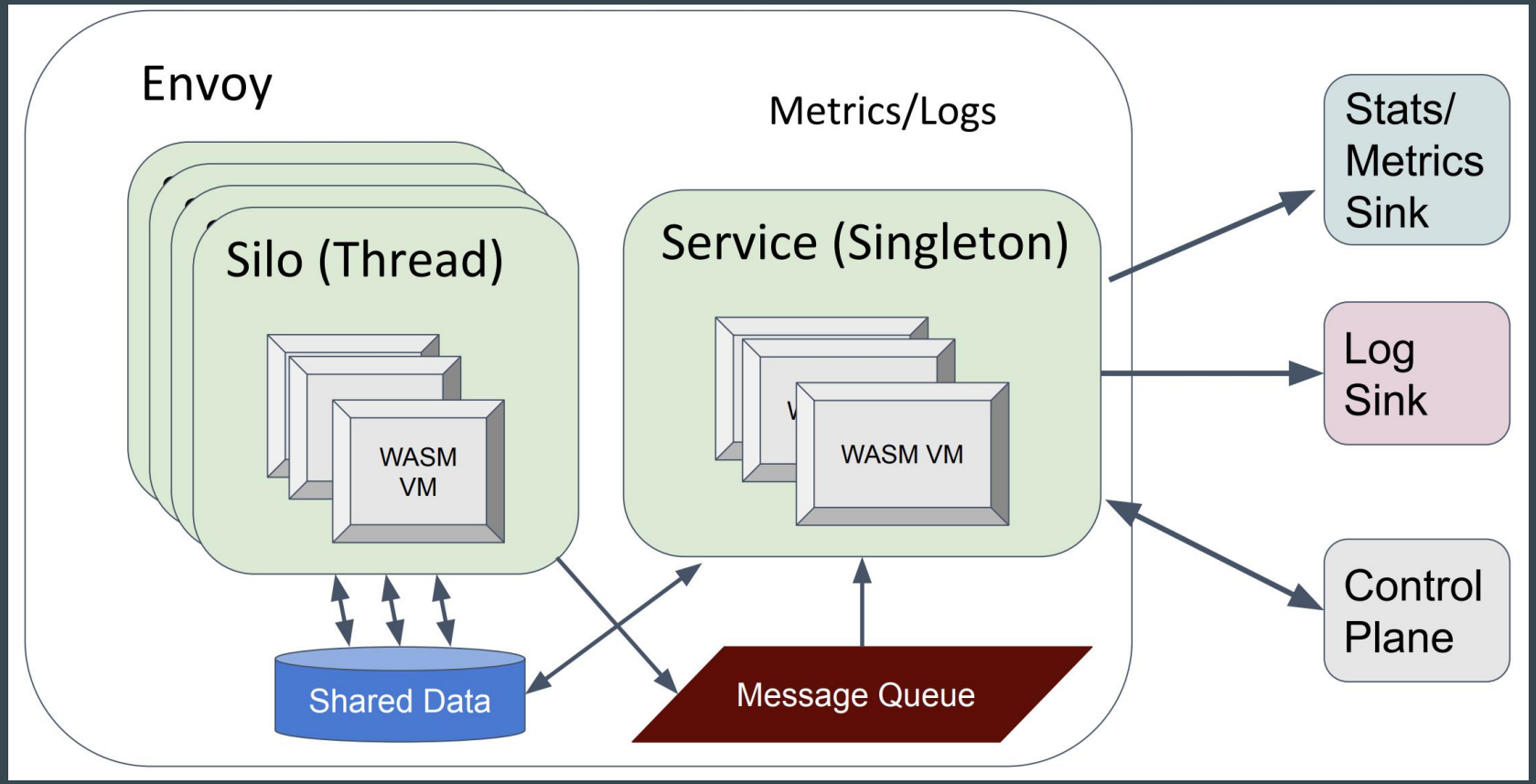
大部分 Envoy 自带扩展都是基于开源的标准，订制的扩展需要

- C++, 编译Envoy
- Lua

Wasm 扩展的优势

- 动态加载
- 可以用多种语言开发
- 性能强于 Lua
- 安全性：沙盒运行
- 可移植到任何实现了所需要的 Applicaiton Binary Interface (ABI) 的环境

Wasm 扩展架构


Envoy
Metrics/Logs
Silo (Thread)
WASM VM
Service (Singleton)
WASM VM
Stats/ Metrics Sink
Log Sink
Control Plane
Shared Data
Message Queue

Wasm 扩展加载过程

```
{
  "name": "my-filter",
  ...
   "config": {
    "vm_config": {
     "vm_id": "test-vm",
     "runtime": "envoy.wasm.runtime.v8",
     "code": {
      "local": {
       "filename": "my-extension.wasm"
      }
     }
    },
    "configuration": {
     "@type": "type.googleapis.com/google.protobuf.StringValue",
     "value": "...."
    }
   }
  }
}
```

1) 当 Envoy 主线程 加载带有 Wasm 扩展的 listener 时，开始创建一个 V8 Isolate，并且将 Envoy 对于 Wasm ABI 实现注入到创建的 Context
2) V8 加载，解析，编译 Wasm module
3) Envoy Wasm runtime 从 V8 Isolate 中读取扩展实现

至此，主线程完成创建 Base Wasm 实例 (模板 Wasm 实例)

1) 模板 Wasm 实例克隆到各个 工作线程
2) 各个线程的 Wasm 实例读取配置完成初始化

Wasm ABI/APIs -- Envoy 内的实现

- **Network filters**: accept/reject connection, get/set metadata, get/set payload, logs.
- **HTTP filters**: accept/reject request, get/set metadata, get/set headers, get/set body, logs.
- **Side calls**: HTTP, gRPC.
- **Logging, Stats**.
- **Shared key-value store.**
- **Message queue.**

Wasm ABI/APIs -- Wasm module 内的实现

- **HTTP**: on header, on body, on trailers
- **TCP**: on connection, on data
- **Timer**: on tick
- **Side calls**: on gRPC/HTTP call success/fail/cancel

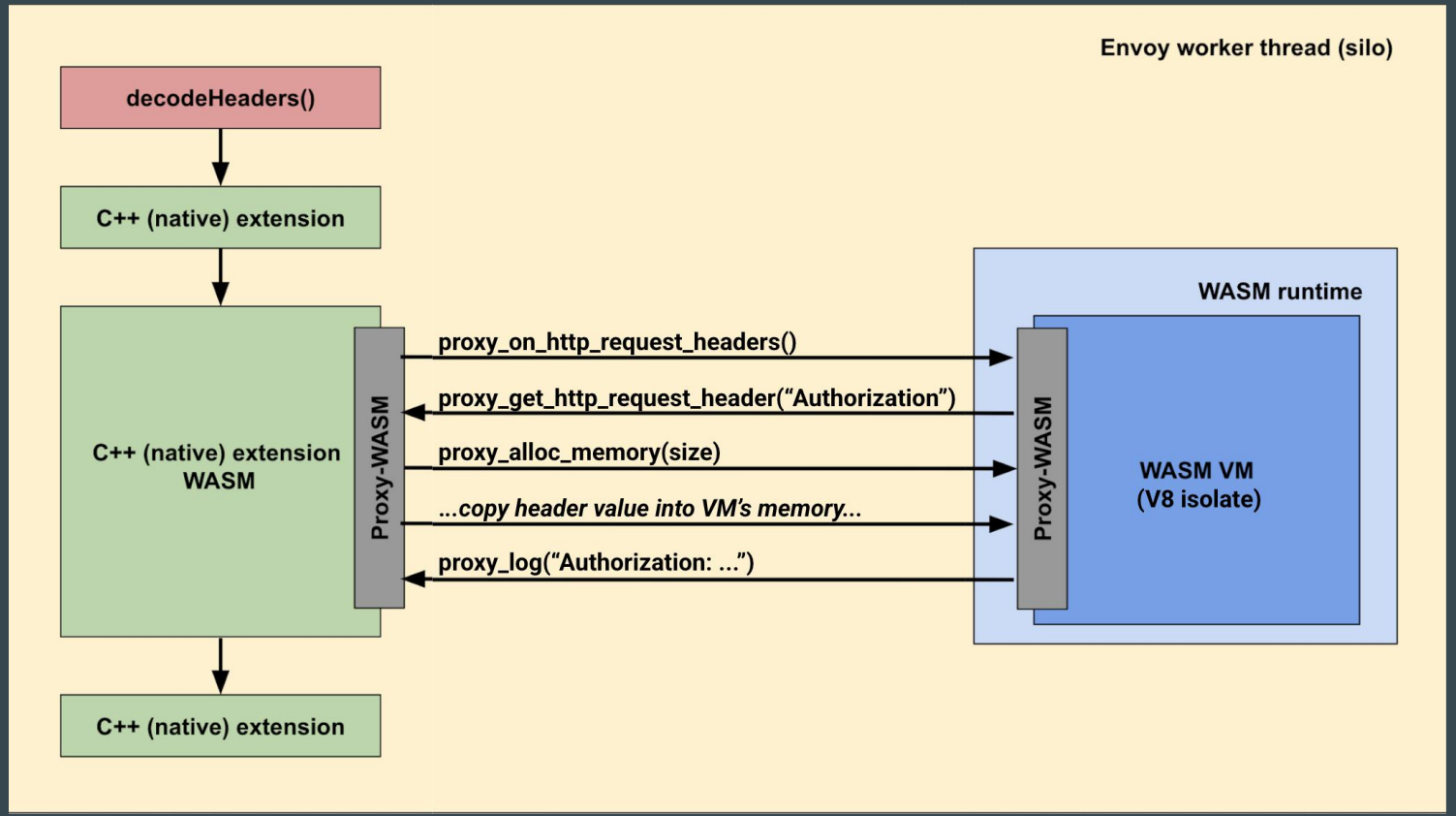
示例 C++ 实现

```
1.  #include "envoy_wasm_filter_intrinsics.h"
2.
3.  class ExampleContext : public Context {
4.    explicit ExampleContext(uint32_t id) : Context(id) {}
5.    FilterHeadersStatus onRequestHeaders() override;
6.  }

7.  FilterHeadersStatus ExampleContext::onRequestHeaders() {
8.    auto result = getRequestHeader("Authorization");
9.    logInfo(std::string("Authorization: ") + std::string(result->view()));
10.   return FilterHeadersStatus::Continue;
11. }

12. static RegisterContextFactory register_ExampleContext(CONTEXT_FACTORY(ExampleContext));
```

Wasm 扩展工作流程


Envoy worker thread (silo)
decodeHeaders()
C++ (native) extension
C++ (native) extension
WASM
Proxy-WASM
proxy_on_http_request_headers()
proxy_get_http_request_header("Authorization")
proxy_alloc_memory(size)
...copy header value into VM's memory...
proxy_log("Authorization: ...")
WASM runtime
Proxy-WASM
WASM VM
(V8 isolate)
C++ (native) extension

性能开销

- 内存
  - Wasm module 代码: 占用 xxx Kb 到 x Mb 不等
  - Wasm 运行时的线性内存
  - V8 自身运行内存：~ 10 Mb
- CPU
  - V8 背景线程
  - C++ 到 Wasm 之间的切换

Istio 配置 Wasm filter

- 通过 EnvoyFilter 配置

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
...
  - patch:
      operation: INSERT_BEFORE
      value:
        name: istio.stats
        typed_config:
          '@type': type.googleapis.com/envoy.extensions.filters.http.wasm.v3.Wasm
          value:
            config:
              configuration:
                '@type': type.googleapis.com/google.protobuf.StringValue
                value: |
                  { ... }
              vm_config:
                code:
                  remote:
                    http_uri:
                      uri: https://xxx.wasm
                      cluster: storage.googleapis.com
                      timeout: 10s
                    sha256: ac7c06d28384a19e74e2.....
                runtime: envoy.wasm.runtime.v8
                vm_id: stats_outbound
```

Istio 发布 Wasm module

- Remote load 问题：和 xDS 分离
- Istio 1.9 解决方案：Istio Agent 截取 xDS 更新，下载 Wasm module 文件，重写 remote load 为本地文件

- 未来：Container Storage Interface (CSI) driver daemonset

未来工作

- Exception 处理
- Crash 处理
- WASI (WebAssembly System Interface)
- 性能提升 (和 V8 合作)
- 执行第三方不可信的 Wasm 扩展
  - Envoy 可以控制 Wasm 资源（内存和CPU）以及 ABI/API 的使用