platformatory
Cloud native engineering studio

Envoy + WASM

Presented at Docker Meetup – 19 Aug 2023

Today's story

platformatory
Cloud native engineering studio

- ➔ A tale of proxies
- ➔ A future of standardization of proxy plugins
- ➔ Demo

About me

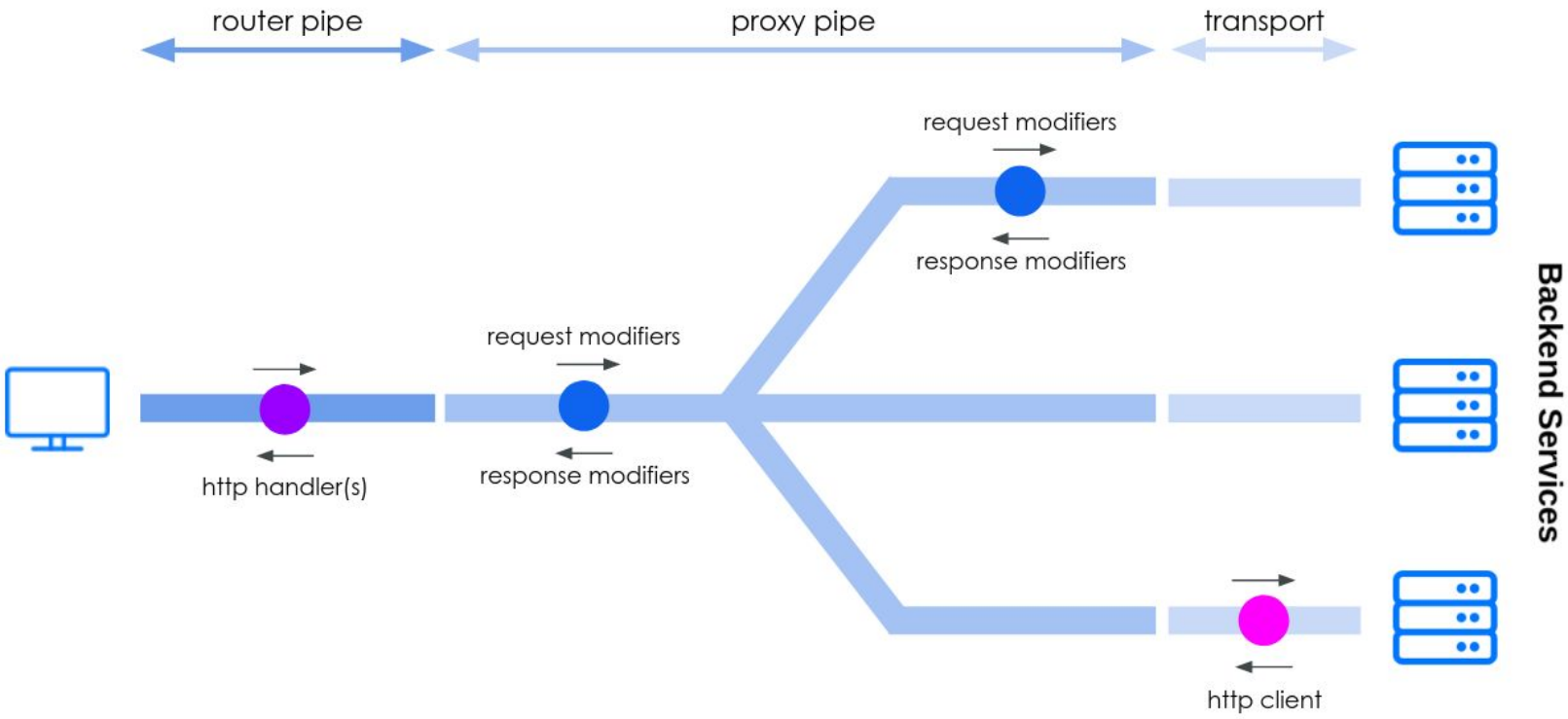
- Platform Engineer @ platformatory.io
- Local meetup organizer for Kong, Grafana, Docker, Kafka and Bangalore Streams
- OSS contribs in Cloud Native space

platformatory
Cloud native engineering studio

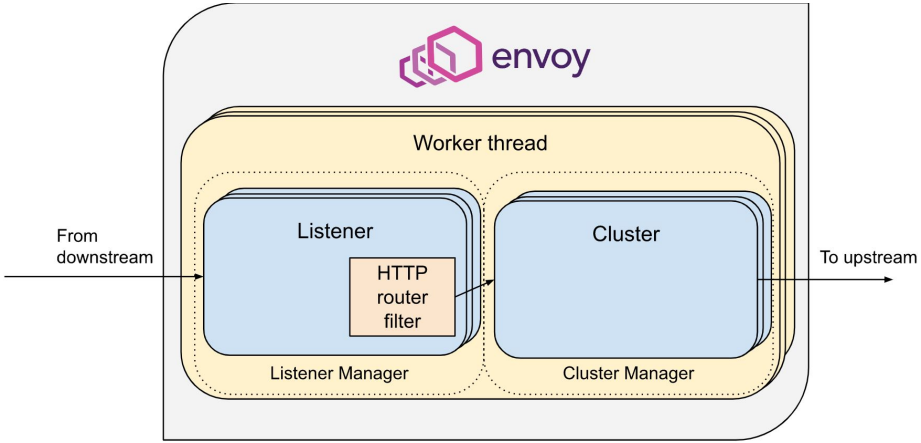

@avinash_ukr


avinash-upadhyaya


avinashupadhya99

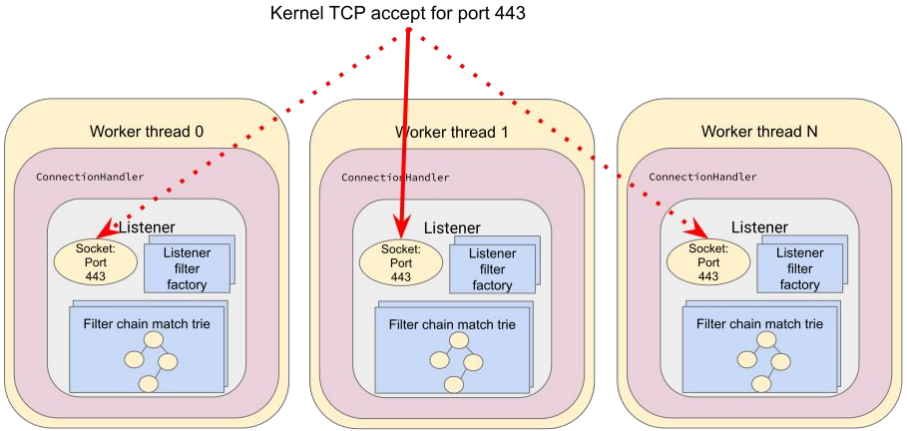


API Gateways = A world of proxies

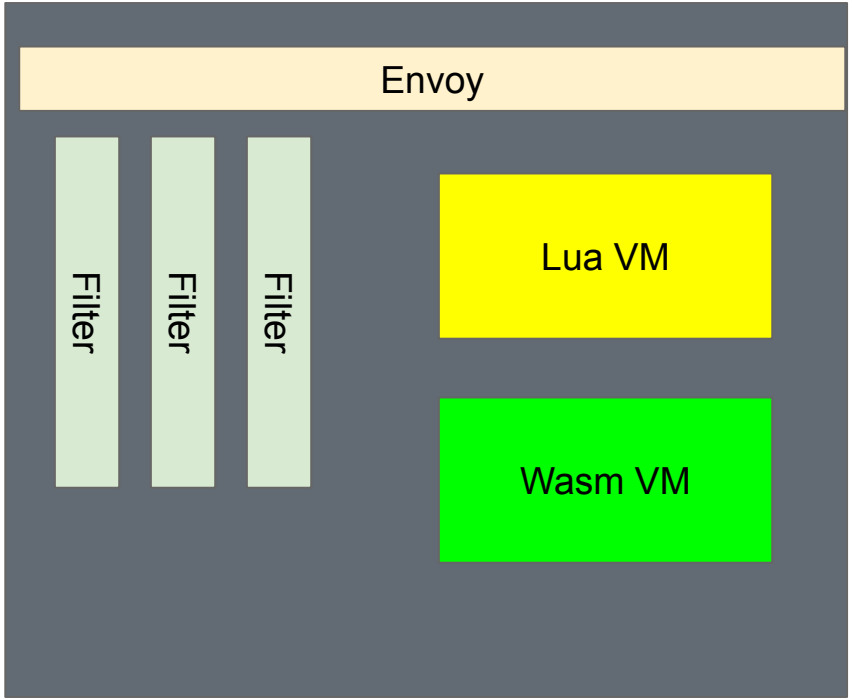


- Born @ Lyft
- Written in C++
- High performance L4-L7 Interception
- A ton of capabilities
  - HTTP/2, gRPC
  - Service Discovery
  - Zone-aware load Balancing
  - Observability
  - ..and much more
- **Extendable, Programmable**
- Ideal for light-weight out of process (typically sidecar container) to handle all network concerns

A word about Envoy: De-facto data plane of the cloud native world



- Implement custom logic within the proxy
- Chained
- Asynchronous I/O
- Coupled with envoy binary



WebAssembly in Envoy

Background

As of early 2019, Envoy is a statically compiled binary, with all its extensions compiled at build-time. This means that projects that provide custom extensions (e.g. Istio), must maintain and distribute their own binaries, instead of using the official and unmodified Envoy binary.

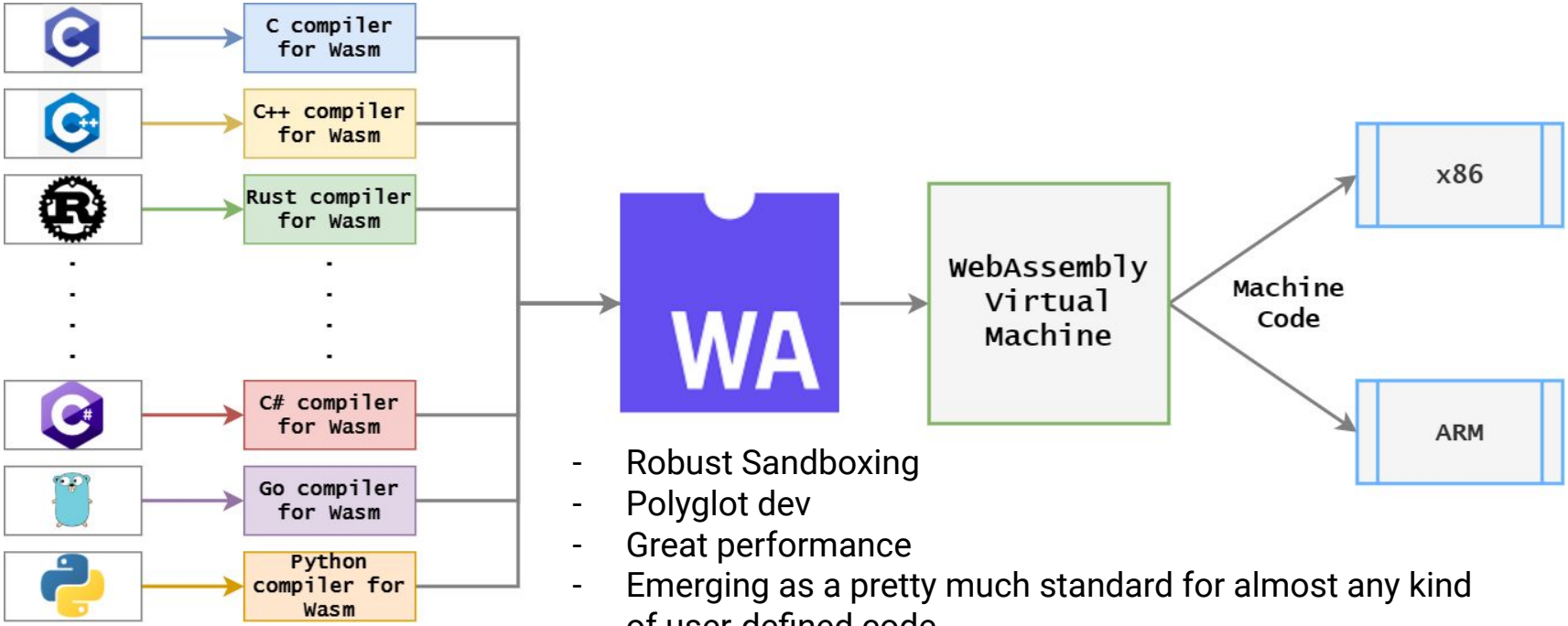
For projects that don't control their deployment, this is even more problematic, because any updates and/or bug fixes to extensions require building new binary, producing release, distributing it, and more importantly, re-deploying it in production.

This also means that there is often a version drift between deployed extensions and the control plane configuring them.

Solution

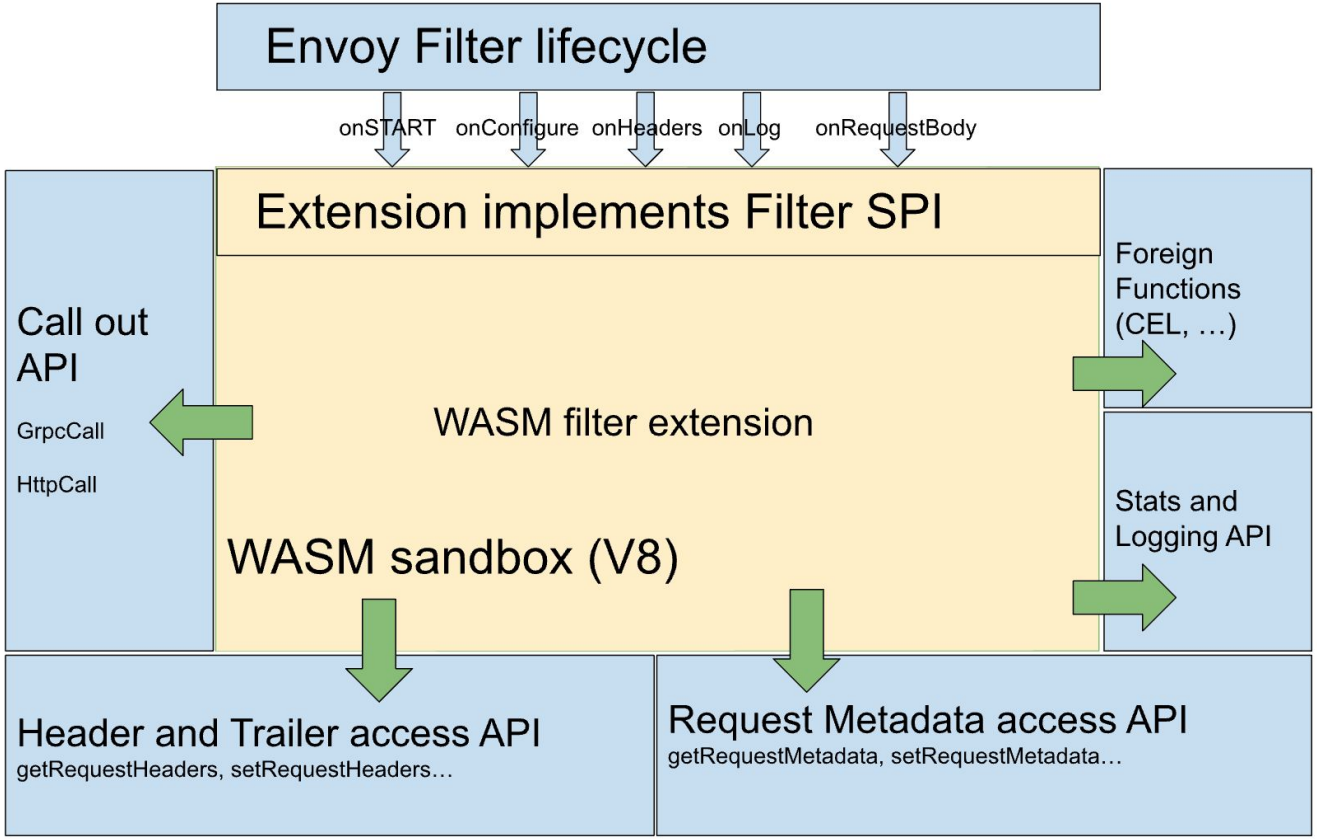
While parts of the issue could be solved using dynamically loadable C++ extensions, this isn't feasible solution at this time, because, due to the rapid pace of Envoy development, there is no stable ABI, or even API, for the extensions, and more often than not, updating Envoy requires code changes, which makes updating a manual process.

Instead, we've decided to address this problem by writing and delivering Envoy extensions in WebAssembly with stable ABI, since it brings many additional benefits (described below).

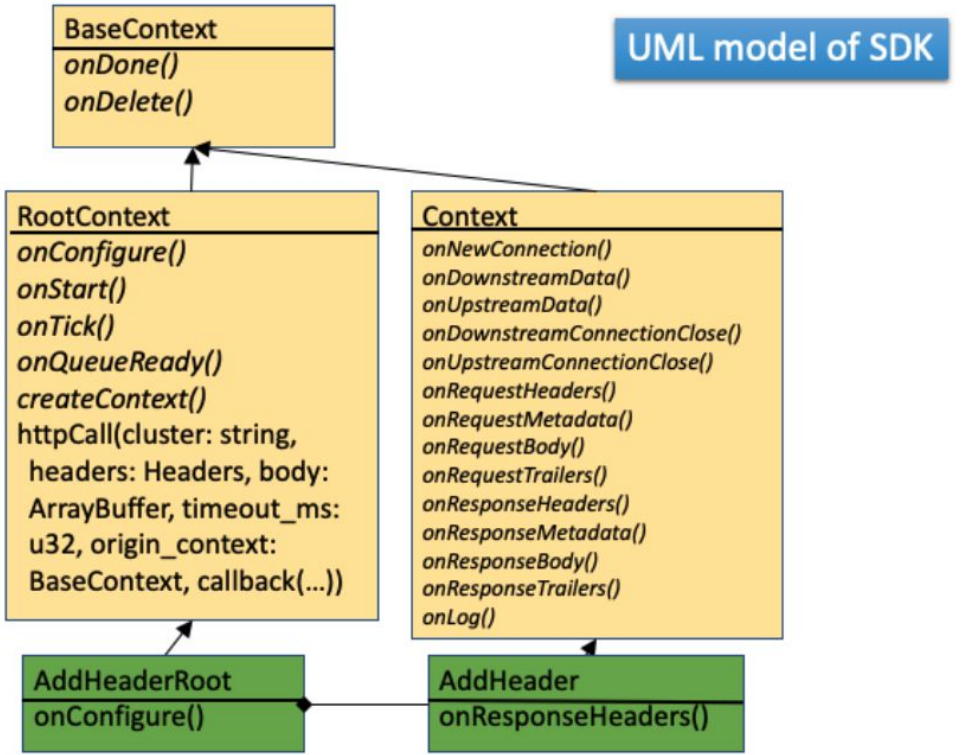


- Robust Sandboxing
- Polyglot dev
- Great performance
- Emerging as a pretty much standard for almost any kind of user-defined code
- Load filters in runtime
- Stateful operations with WASM Service

Why WASM?



How WASM fits into Envoy




```
registerRootContext(
    () => { return RootContextHelper.wrap(new AddHeaderRoot()); },
    "add_header");
```

Ref:
https://docs.eupraxia.io/docs/how-to-guides/deploy-rust-based-envoy-filter/

Conclusion

platformatory
Cloud native engineering studio

- The future for building proxy plugins = WASM.
- Envoy supports
  - V8
  - wamr
  - wasmtime
  - wavm
- Vast interoperability between other gateway proxies, mesh data planes
  - Appropriate tons of goodness already available in the Envoy ecosystem
  - @see: webassemblyhub
- The Proxy wasm spec is maturing
  - Not everything works. YMMV
  - Performance is dependent on the Wasm Runtime
  - Debug
  - Tooling will get better over time
- You can try this out
  - https://github.com/tetratelabs/proxy-wasm-go-sdk
  - https://github.com/proxy-wasm/proxy-wasm-rust-sdk
- Other Envoy references
  - https://tetrate.io/blog/wasm-modules-and-envoy-extensibility-explained-part-1/

<<DEMO>>

Thank you.