Making Envoy Contributions Feasible for Everyone

Yaroslav Skopets, Tetrate.io

Envoy TLDR

- a *network proxy*
- that runs *alongside* applications
- to provide them with *common features*
- in a *platform agnostic* manner

Envoy Insight

- *can see every request* received or made by application

Envoy Opportunity

Could we reuse Envoy ?

- to *learn* from the *actual traffic*
- *efficiently*
- *flexibly*
- *easily*

Envoy Extensibility (Native)

Native Envoy extensions:

- developed in *C++*
- *statically* linked into Envoy binary
- imply *custom builds* of Envoy
- overall, a lot of *investment* and *commitment* *upfront*

Can we do better ?

WebAssembly TLDR

- *low-level* code format
- for *safe* and *efficient* execution
- in a *sandboxed* environment

WebAssembly USE

- applications developed using a *regular* programming language
- but *compiled into* WebAssembly code

Envoy Extensibility (WebAssembly)

+

WebAssembly-based Envoy extensions:

- developed in *Rust*, *AssemblyScript*, *(Tiny)Go*, *C/C++*, *etc*
- *loaded/unloaded* on demand
- offer *less capabilities* than native selves

Sounds interesting. Let’s give it a try!

Challenge

OPEN
{ API }

Context:

- microservice-based architecture
- *REST API* + *OpenAPI Specs*
- *Contract First* API Development

Questions:

- Are *API specs complete* and *up-to-date* ?
- Do *implementations match API specs* ?

Experiment

OPEN
{ API }

Requirements:

1. *validate* requests against *API Spec*
2. *make* violations *noticeable* (metrics)

Let's develop an Envoy extension for that:

- *ad-hoc*
- *experimental*
- *disposable*

AssemblyScript TLDR

- *subset* of *TypeScript syntax*
- *statically typed*
- *garbage collected*
- *compiled into WebAssembly* code

AssemblyScript TLDR

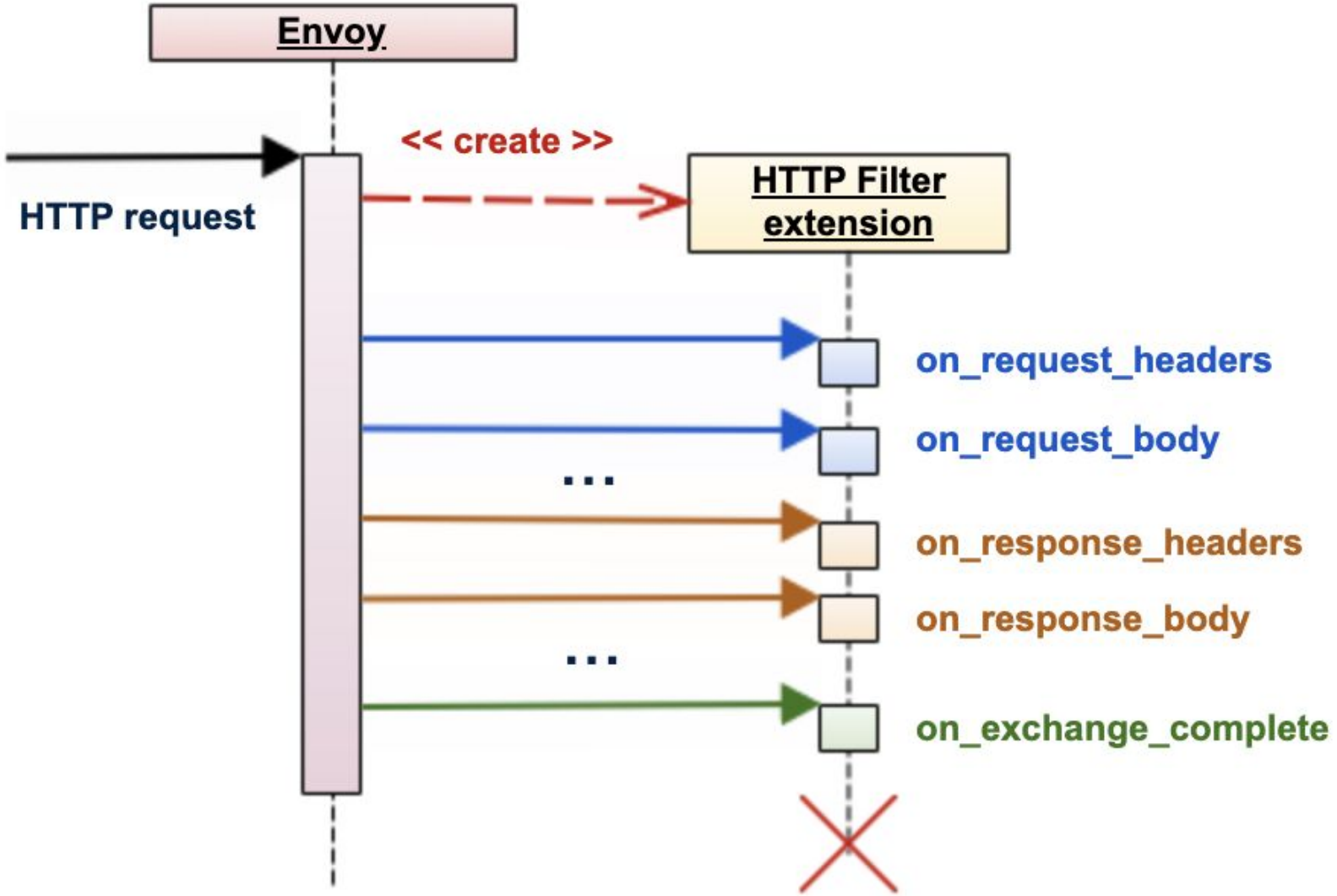
- NOT a *JavaScript*
- CAN'T reuse *JavaScript* libraries
- MIGHT reuse *some* *TypeScript* libraries

Overall, *feels like* *TypeScript*

AssemblyScript WHY

- *complexity* of the solution == complexity of the problem
- *familiar* syntax
- *familiar* toolbox (*npm*)
- *productive* development cycle

Extension Model



Walkthrough: Getting Started


```
class ApiValidator extends HttpFilter {
 constructor() { super(); }
}
```

Walkthrough: Data model


```
class ApiSpec {
 operations: Array<Operation>
}

class Operation {
 method: string
 path:   string
}
```

Walkthrough: Configuration


```
class ApiValidator extends HttpFilter {
 private spec: ApiSpec

 constructor(config: string) {
   super();
   this.spec = ApiSpec.parse(config);
 }
}
```

Walkthrough: Request validation


```
class ApiValidator extends HttpFilter {
  onExchangeComplete(): void {
    let method = context.getRequestHeader(":method");
    let path   = context.getRequestHeader(":path");

    if (!this.spec.contains(method, path)) {
      log.warn("unknown API: " + method + " " + path);
    }
  }
}
```

Walkthrough: Test

Expected Envoy output:

```
...
starting main dispatch loop
...
wasm log api_validator : unknown API: GET /orders
wasm log api_validator : unknown API: GET /orders/1
wasm log api_validator : unknown API: GET /orders/1/items
wasm log api_validator : unknown API: GET /orders/1/items/2
...
```

Walkthrough: Metrics


```
class ApiValidator extends HttpFilter {
  onExchangeComplete(): void {
    ...
    if (!this.spec.contains(method, path)) {
      Stats.counter("api_validator.violations_total").inc();
    }
  }
}
```

Walkthrough: Test

Expected Envoy stats:

```
$ curl -s http://localhost:9901/stats | grep api_validator


api_validator.violations_total: 4
```

Envoy Community

- give it a *try*
- share *feedback*
- join us to build idiomatic *Envoy SDKs* for *Rust*, *AssemblyScript*, *(Tiny)Go*, etc

Resources

Source code:
https://github.com/yskopets/kubecon2020

Envoy w/ WebAssembly support:
https://github.com/envoyproxy/envoy-wasm

Envoy WebAssembly ABI:
https://github.com/proxy-wasm/spec


KubeCon
CloudNativeCon
Europe 2020
Virtual
HELM
KEEP CLOUD NATIVE
CONNECTED