KubeCon | CloudNativeCon

Europe 2020

Virtual

Building a Distributed API Gateway with a Service Mesh

Rei Shimizu, Waseda University & Cynthia Coan, Tetrate

Outline / Agenda

What is Service Mesh?

De-Generalizing “API Gateway”

What is Envoy?

Using WASM in Envoy

Demo (WASM + Getenvoy)

Service Mesh 101

Bring Standardization to Networks When Dealing With Services

- Infrastructure Layer designed to help service-to-service communication.
- Designed to help create a separation between the network, and the application logic.
  - Can help enforce policies across the entire network, and let developers change without overloading them in their support of their app.
- Applications have “Sidecars” that handle network communication.
  - Instead of Application A talking to Application B directly, it tells the sidecar it wants to talk to application b.
  - The Application A sidecar then looks up it’s policies that have been configured including route information, security, retry settings, etc. and routes the request.
  - The Application B sidecar can then sees a request coming from Application A, and validates security settings, etc., and routes to Application B.

“API Gateway”

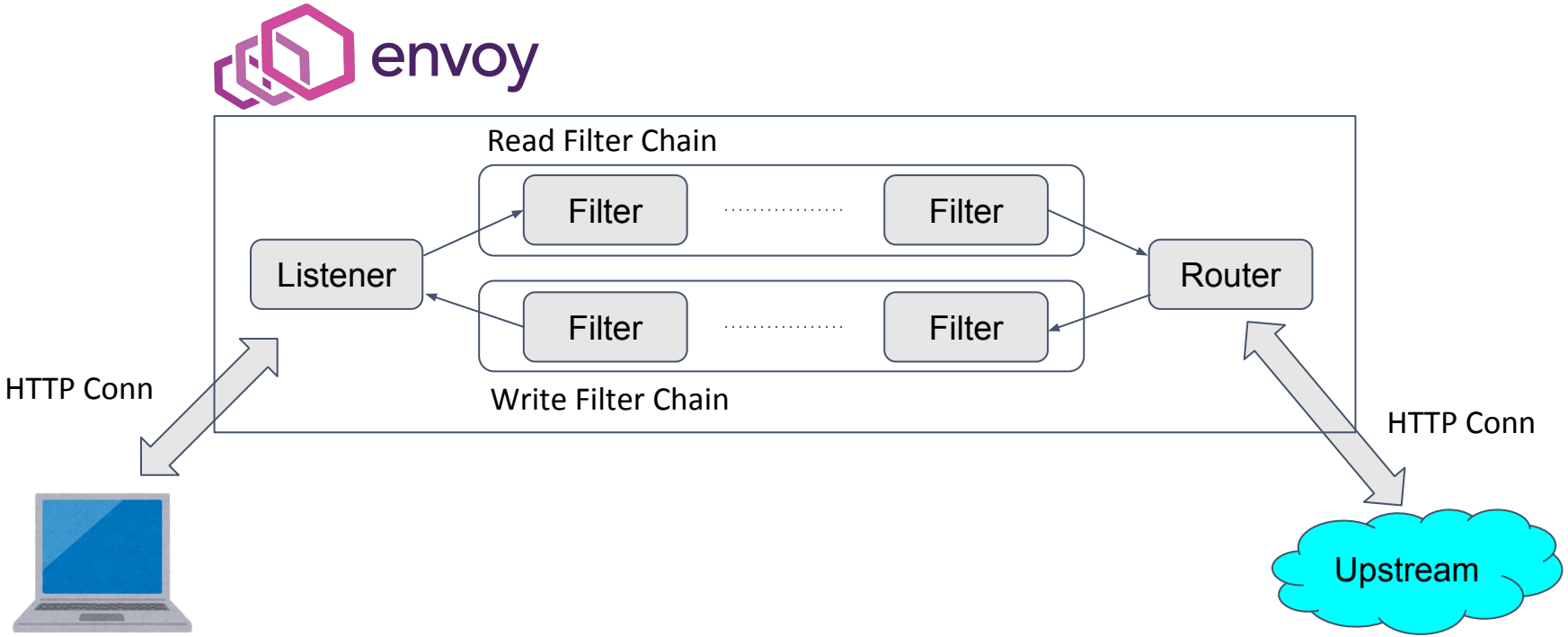
- API Gateway is one of those sets of words where everyone has a slightly different idea of what an API Gateway should actually provide, and is.
- API Gateways are a similar idea to a service mesh but at the “Front Door”.
  - API Gateways are a configurable proxy that sit in front of your services.
  - One of the core things they do very similarly to a service mesh is provide a consistent entrypoint to your applications.
  - Configuring Authentication, Routing, Rate-Limiting, protocol translation, etc. to all the services behind the mesh.
- If the Service-Mesh and API Gateway are doing similar things why can’t we combine these two things together?
  - Rather than maintaining two separate systems that are both doing authentication, routing, etc. why not just maintain one service that does this?
  - There isn’t feature creep as we’re not adding two features, we’re merging things doing the same thing into one place so it’s easier to reason about.

What is Envoy?

Cloud-Native high-performance edge/middle/service proxy

- Proxy for Cloud Native era
- Written in modern C++
- Have a possibility to achieve disentanglement between network and applications
  - Dynamically configurable
  - Extensibility through “Filters” at the L4, and L7 layers
  - First-Class Logging, and Monitoring
  - High-Performance due to lots of optimization, and threading work
- In this session, we use envoy as the sidecar in our service mesh

Basic Architecture



Wasm extensibility for Envoy

- Envoy now has support for extending it via WASM
  - https://github.com/envoyproxy/envoy-wasm
- Official SDKs for C++/Rust.
  - Unofficial SDKs are also available for Go/AssemblyScript (and you can make your own!).
- Wasm filters can be delivered dynamically to Envoy with it's "xDS" configuration system.
  - This makes it possible to update WASM filters with zero downtime.
- Without WASM we'd have to build our extensions into the envoy binary with C++.
  - This means it is impossible to dynamically update a filter in Envoy.
  - You also have to start building your own Envoy.
    - E.g. Istio-proxy is built on the top of envoy with custom filters.
  - It should be mentioned lua is also an extension mechanism supported, but is not full featured.

What is Wasm?

- WebAssembly is constructed to run on the browser, like JavaScript.
  - But, WebAssembly is much faster than JavaScript because that is only simple binary format.
    - That is strictly typed, so that it is easy to optimize.
    - Not needed to parse. lightweight to deliver.
- In addition to this, the runtime is sandboxed.
  - It is because Wasm is constructed to run on the browser.
  - It won't collapse the host environment.

Why Wasm?

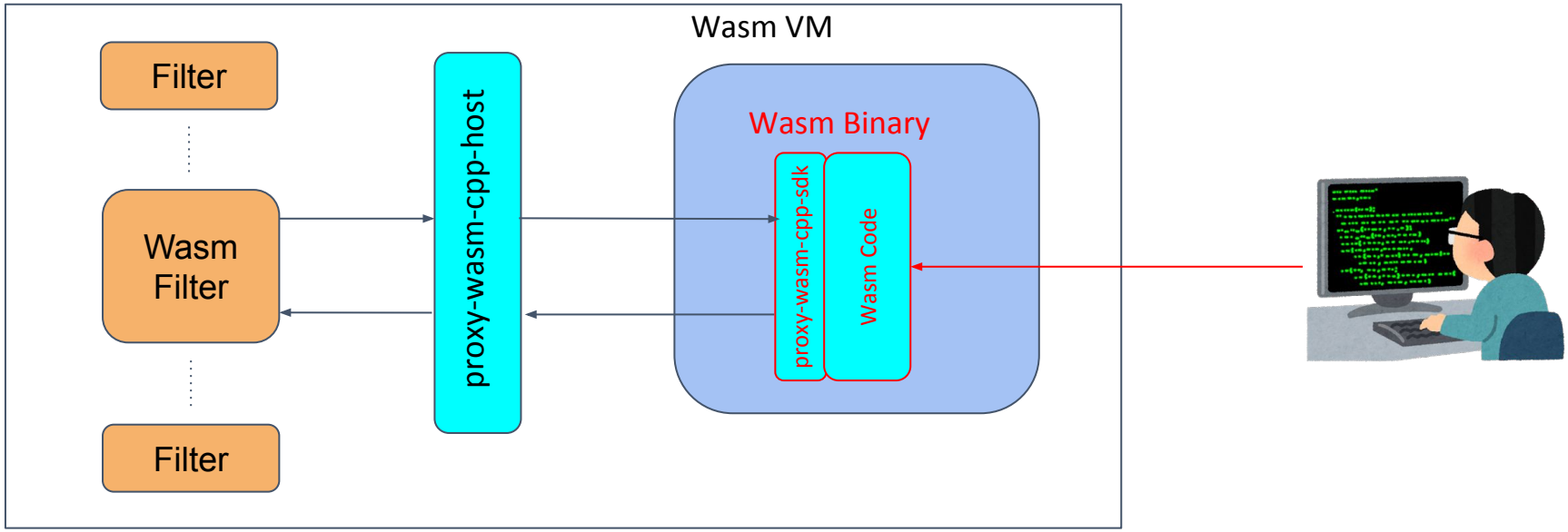
- Isolated Environment
  - Wasm runtime won't collapse the host environment.
  - CPU usage and memory consumption can be limited.
  - Sandboxed runtime can block malicious operations from host environment.
    - It means that we can preserve sensitive informations, such as tokens.
- The number of supporting languages
  - There is many languages that supports Wasm, such as C/C++, Rust, Go, TypeScript etc…
- Portability
  - We can separate between host environment (e.g. Envoy) and runtime (e.g. V8)
  - It means that we can introduce Wasm extensibility to much proxies, such as nginx.

What is proxy-wasm?

- WebAssembly for proxies
  - It means that wasm extensibility is not only for Envoy, but also other proxies.
  - Envoy is one of reference implementation of this.
  - Apache Traffic Server(ATS) also has proxy-wasm implementation.
- We can say that proxy-wasm is the set of specifications of ABI for proxies.

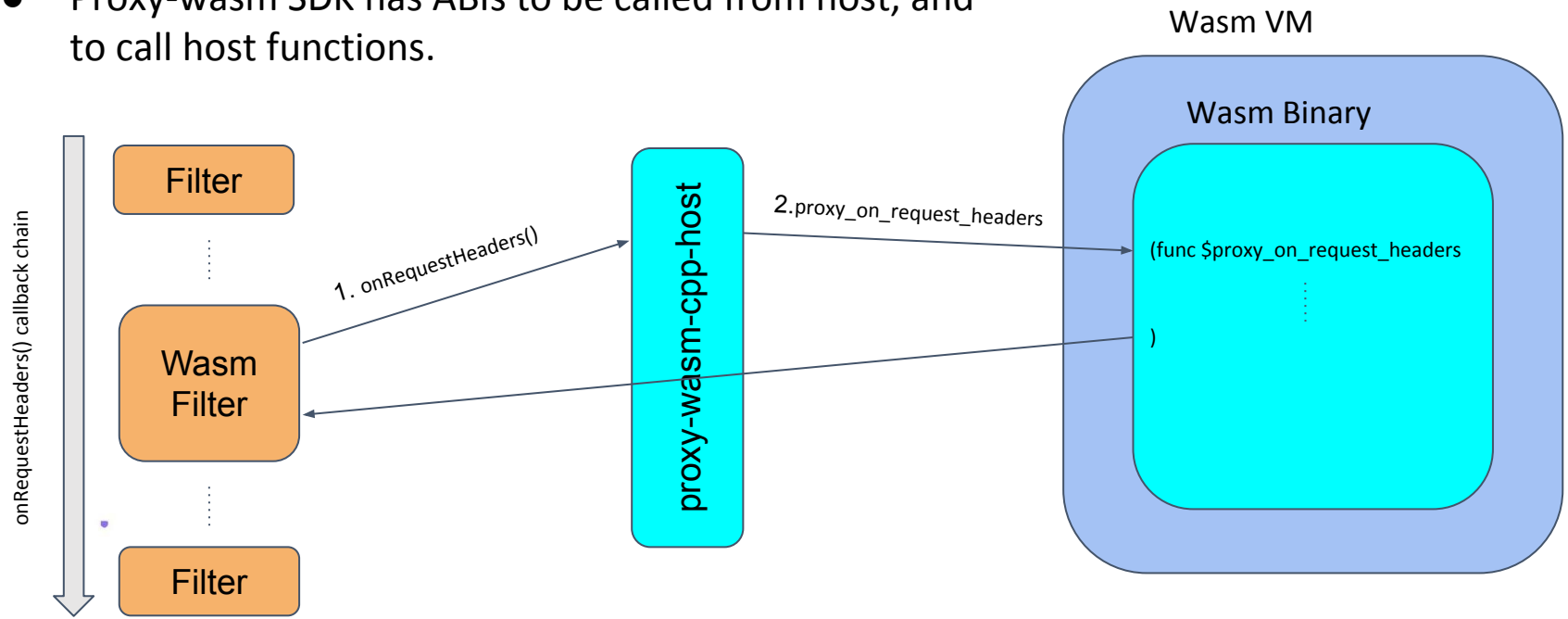
Basic Architecture

Envoy Worker Thread (Silo)



Write wasm filter

- You can write Wasm filter in C++ and Rust.
  - These languages has official proxy-wasm SDK.
- Proxy-wasm SDK has ABIs to be called from host, and to call host functions.



Write wasm filter

- Write Wasm filter for JWT validation in Rust.
  - This code is powered by unofficial (internal-manufactured) Rust SDK.
  - Officially supported to use Rust.

https://github.com/proxy-wasm/proxy-wasm-rust-sdk

- We can write with Host Functions to call exposed Envoy functions.

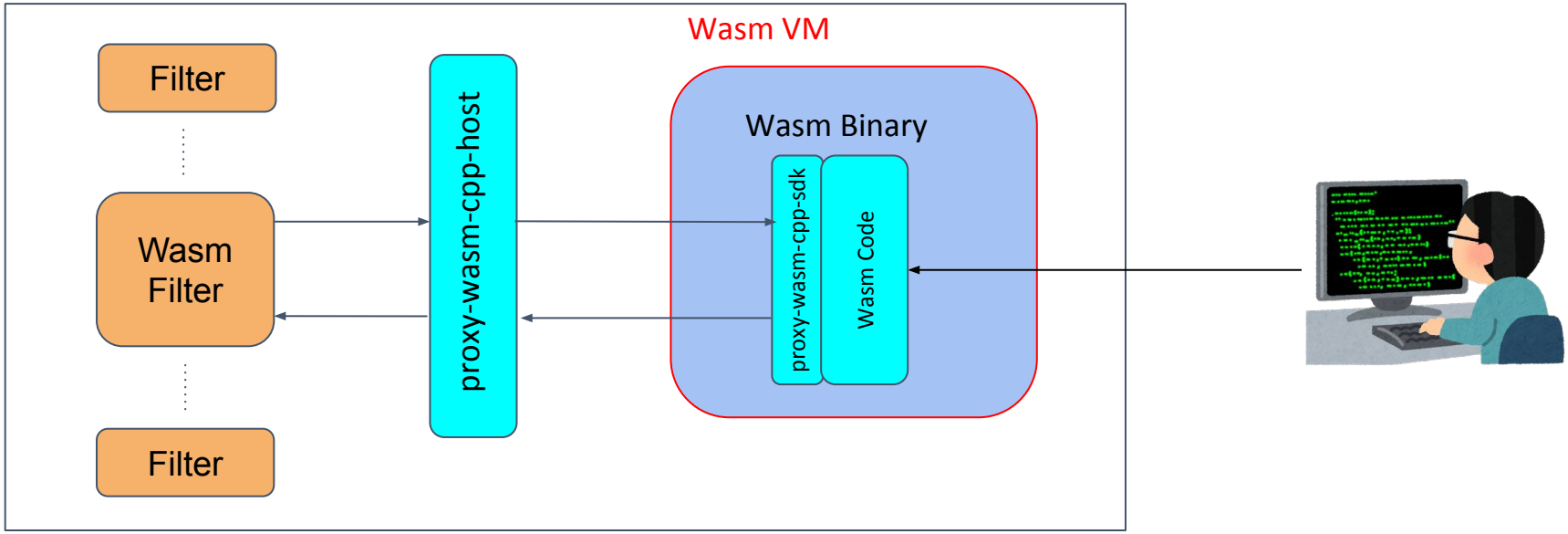
```rust
fn on_request_headers(&self, _headers: u32) -> FilterHeadersStatus {
    // JWT restriction path matcher
    let path_matcher = Regex::new(r"/.*/private$").unwrap();
    let path = get_request_header(":path".to_string()).unwrap().to_string();
    if path_matcher.is_match(&path.as_str()) {
        let data = get_request_header("Authorization".to_string())
            .unwrap()
            .to_string();
        let auth: Vec<&str> = data.split(" ").collect();
        if auth.len() != 2 || auth[0] != "Bearer" || !validate(auth[1]) {
            send_local_response(
                401,
                "".to_string(),
                "Invalid Token\n".to_string(),
                &HashMap::new(),
                GrpcStatus::Ok,
            );
            return FilterHeadersStatus::StopIteration;
        }
    }

    FilterHeadersStatus::Continue
}
```

Host Function

Basic Architecture

Envoy Worker Thread (Silo)



Wasm VM

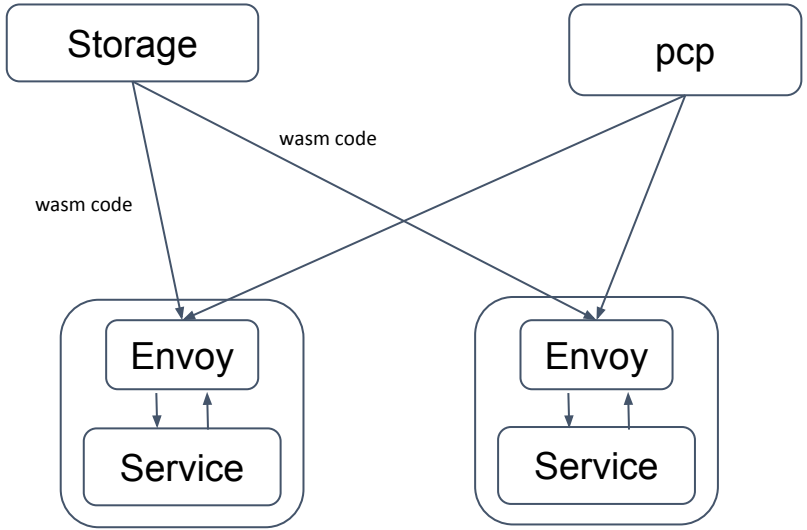
- Wasm code is executed on Wasm runtime, which runs on per the Envoy Worker Thread (Silo).
- We can use V8 and WAVM on Envoy.
- Switching wasm runtime with bootstrap config.

```
{
  "name": "envoy.filters.http.wasm",
  "typed_config": {
    "@type": "type.googleapis.com/envoy.extensions.filters.http.wasm.v3.Wasm",
    "config": {
      "root_id": "my_root_id",
      "vm_config": {
        "runtime": "envoy.wasm.runtime.v8",
        "code": {
          "local": {
            "filename": "./config/lds/envoy_wasm_demo.wasm"
          }
        }
      }
    }
  }
},
```

How to extend our Service Mesh with Wasm?

- We can specify wasm codes from Control Plane via LDS.
  - I developed a simple control plane for delivering wasm code to all of data planes for this demo, called pcp.
  - Assumed to share docker volume in running service containers.
  - NOT production ready.

https://github.com/Shikugawa/pcp



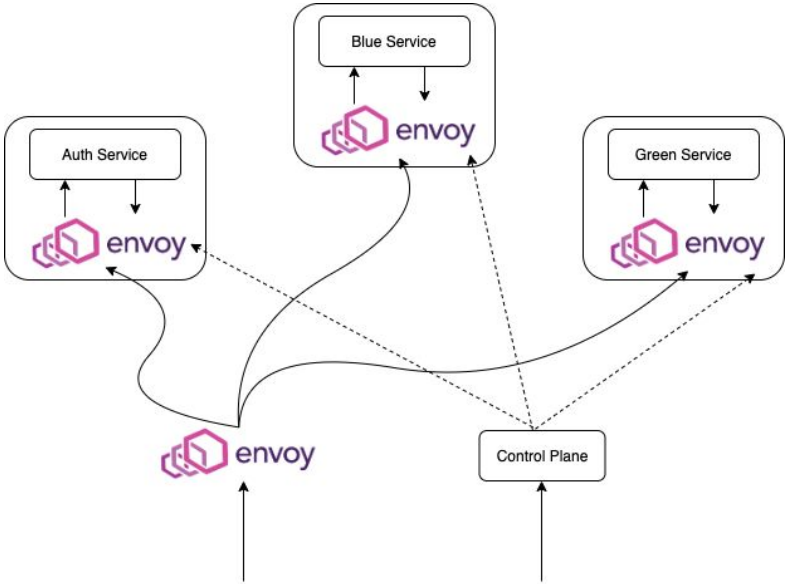
GetEnvoy

- GetEnvoy is a CLI tool developed by Tetrate.
- Provide the easiest way to get envoy binary.
  - It is very useful to verify your envoy environment.

getenvoy run standard:1.15.0 -- --config-path /path/to/config.yaml

Provide AuthN layer with Wasm Filter

- In today's demo, we provide the API Gateway ability via envoy wasm extensibility.
  - By applying AuthN layer to services which constructed on the top of Service Mesh.
  - Apply Basic AuthN and JWT validation filter written in Rust and internal-manufactured Rust SDK.
- Front envoy and service envoy is powered by GetEnvoy.



KubeCon

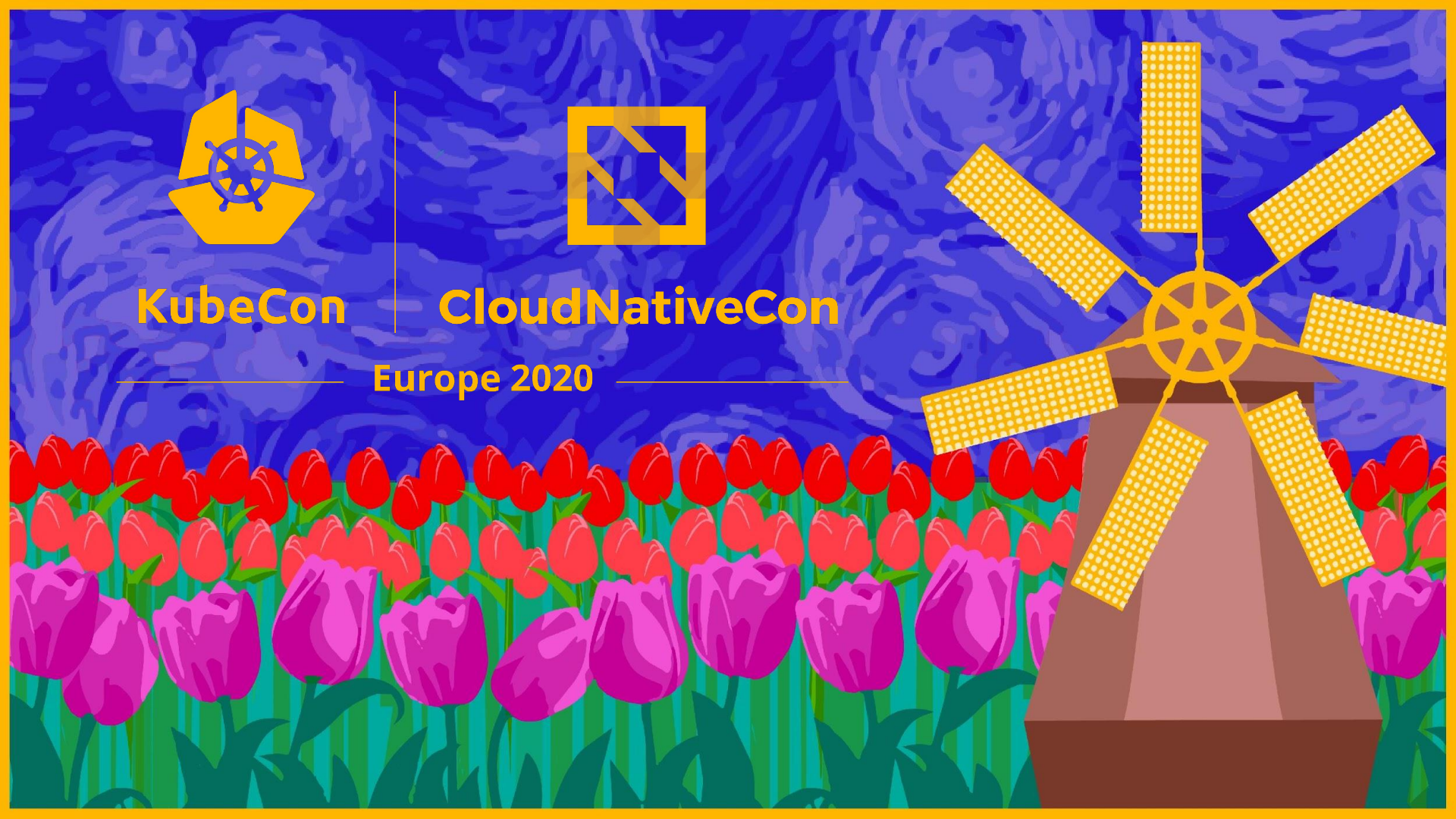
CloudNativeCon

Europe 2020

Demo

GetEnvoy Wasm

- GetEnvoy has also great abilities to accelerate our WebAssembly filter development.
  - Boilerplate with internal-manufactured proxy-wasm SDK
    - Currently it supports only Rust.
    - The internal-manufactured proxy-wasm SDK is based on official proxy-wasm SDK, with some conveniences.
  - Build and test them with docker container.
  - Run WebAssembly filter with envoy by the easiest way.


KubeCon
CloudNativeCon
Europe 2020