
cilium
envoy
KubeCon
CloudNativeCon
Europe 2019
Transparent
Chaos Testing with Envoy,
Cilium, and eBPF
Thomas Graf, Cilium
envoy

The Stack we will build

Go extensions on top of Envoy for rapid development, customization and additional protocol parsers.

Efficient and stable edge/sidecar proxy providing load-balancing, retries, authentication, and much more.

eBPF-powered networking, security, encryption, load-balancing and servicemesh acceleration.

Highly efficient sandboxed virtual machine to execute logic on events inside the Linux kernel.

What is Chaos Testing?

Chaos engineering is the discipline of experimenting on a software system in production in order to build confidence in the system's capability to withstand turbulent and unexpected conditions.[1]

What is Fault Injection?

In software testing, **fault injection** is a technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths, that might otherwise rarely be followed. It is often used with stress testing and is widely considered to be an important part of developing robust software.[1]

Fault Injection Example

cilium

Normal Operation:

Failure Injection:

Requirements for Failure Injection

- **Proxy**
  - Get control of communication between services
- **Ability to simulate failure**
  - Return “504 Application Error” even if service returned “200 OK”
  - Delay response by N msec
- **Probability aspect**
  - Inject failure in 50% of all attempts
- **Transparency**
  - Avoid need to change or restart services
- **Visibility**
  - Was the failure simulated or real?

Let's meet those requirements

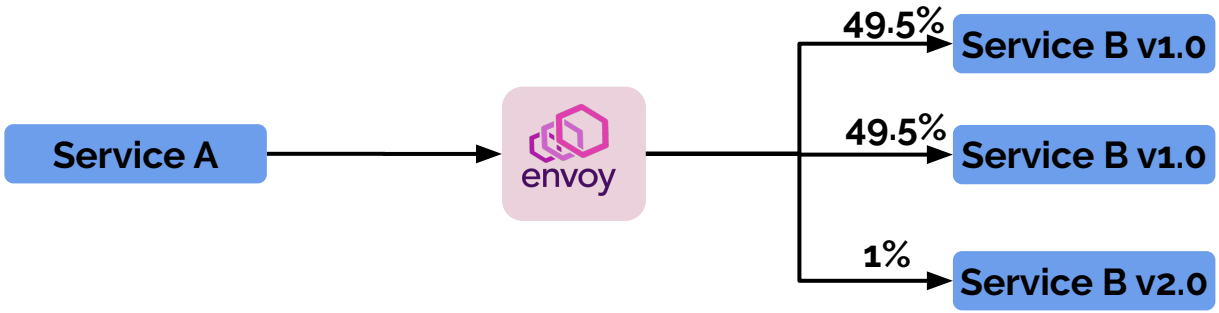
Envoy

- Service and edge proxy
  - HTTP/2, gRPC, MongoDB, DynamicDB with more protocol support coming
- Advanced load-balancing
  - L7, Canary, Retries, Circuit breaking, Rate limits
- Security
  - Authorization, mTLS
- Observability
  - Tracing & metrics
- Extendable
  - Go extensions, WASM, LUA, ...

Envoy Go Extensions

- Ability to extend Envoy with Go code
- Example:
  - Use `net/http` for HTTP parsing
  - Produces `http.Request` and `http.Response` for arbitrary failure injection
  - **Envoy becomes capable of running any Go code HTTP handler**

A simple Envoy Example



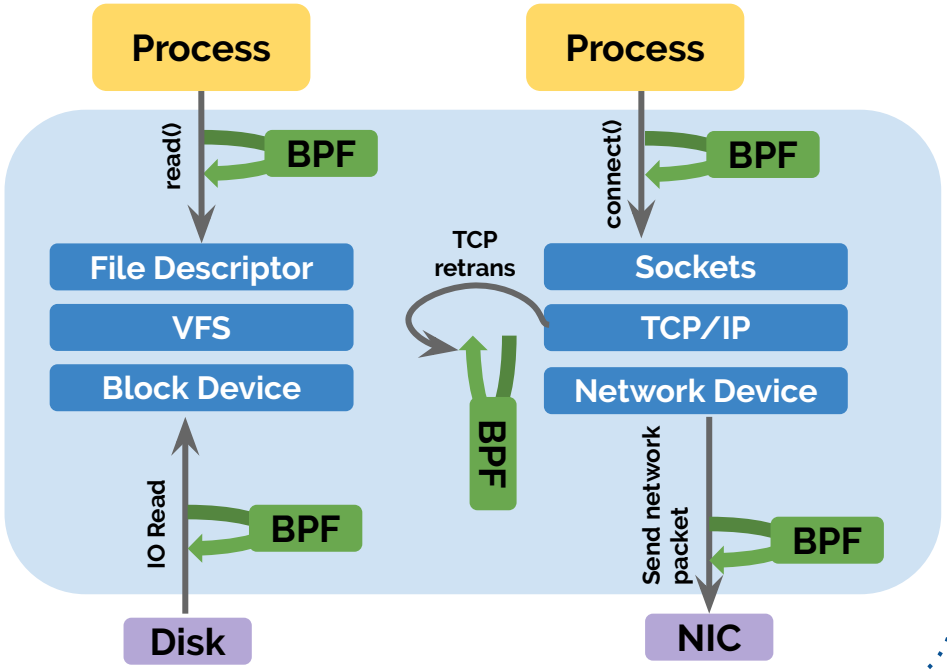
Cilium

- Based on eBPF technology
- Networking
  - Cilium-CNI or chaining on top of most other CNIs
- Kubernetes Services implementation
- Network Policies
  - Identity-based, DNS aware, API aware
- Multicluster, Encryption
- Native Envoy and Istio Integration
  - Transparent Envoy injection (per-node or sidecar)
  - Accelerated proxy redirection, Transparent SSL visibility

eBPF

Highly efficient sandboxed virtual machine in the Linux kernel making the Linux kernel programmable at native execution speed.

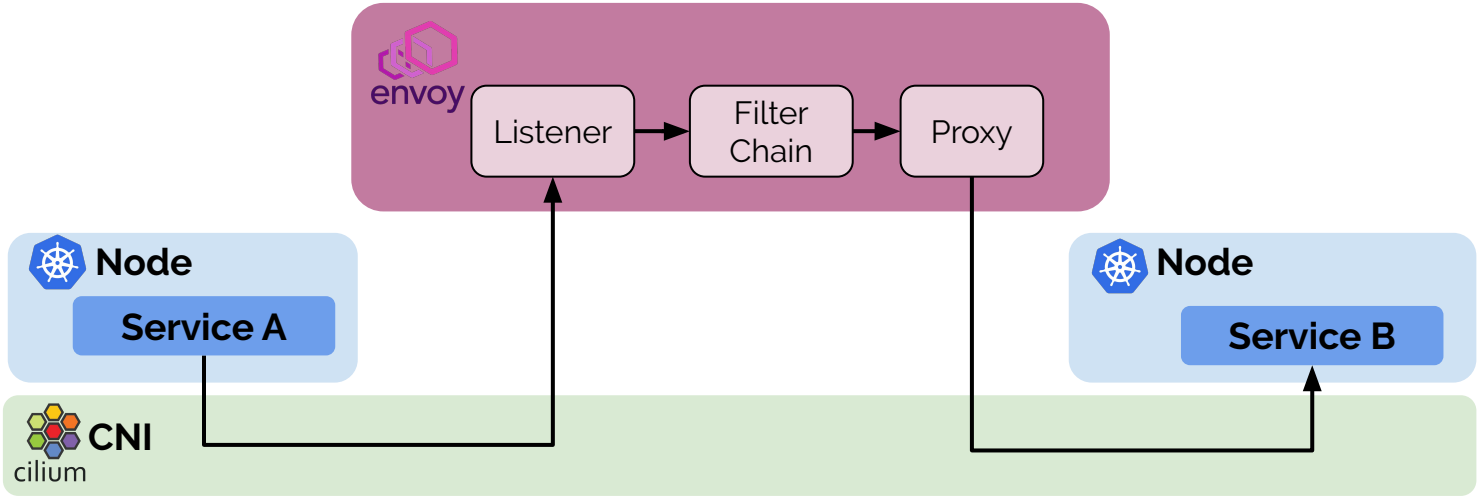
Jointly maintained by Cilium and Facebook with collaborations from Google, Red Hat, Netflix, Netronome, and many others.

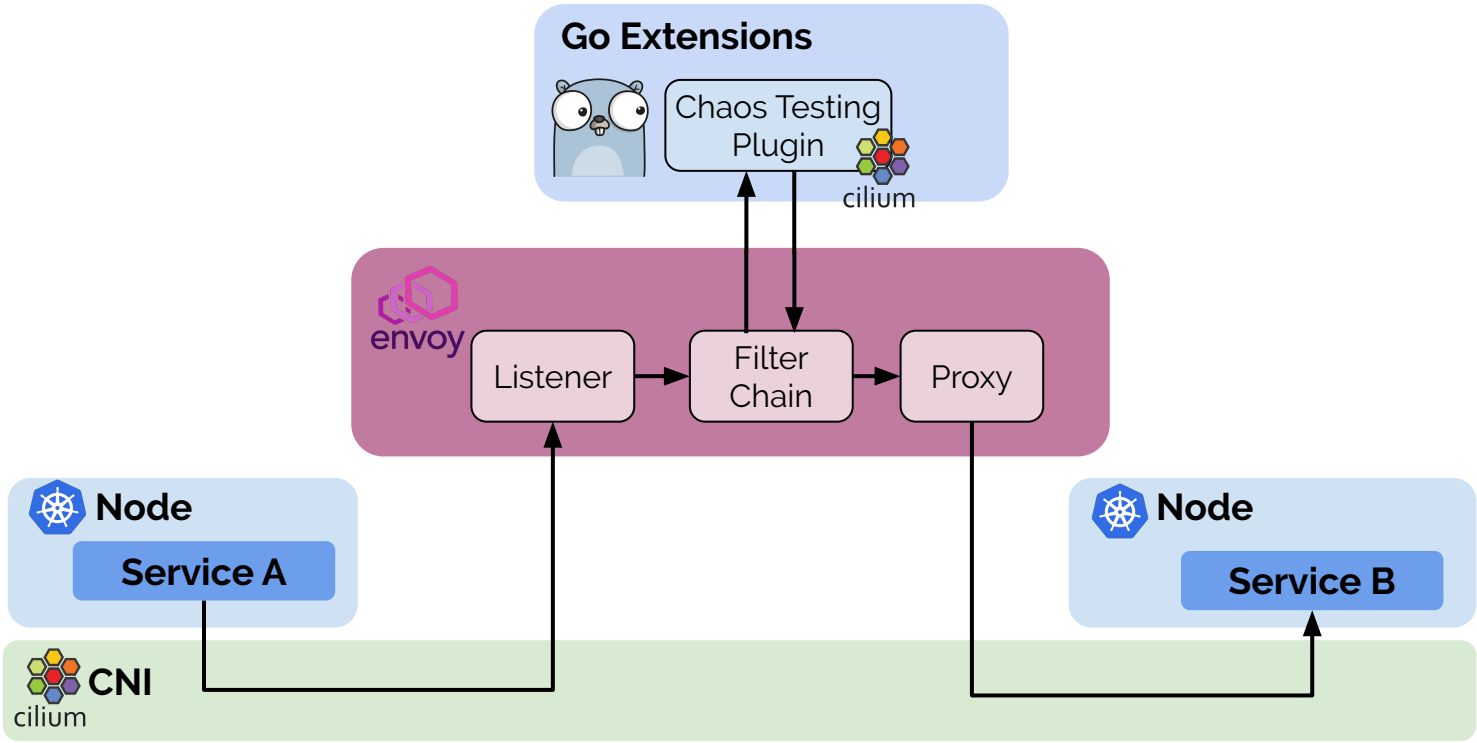


What is transparent chaos testing?


Node
Service A
Node
Service B

cilium


envoy
Listener
Filter Chain
Proxy
Node
Service A
Node
Service B
CNI
cilium


Go Extensions
Chaos Testing Plugin
cilium
envoy
Listener
Filter Chain
Proxy
Node
Service A
Node
Service B
CNI
cilium

Example YAML

Simulate 50% to requests to *myService* on port 8000 to fail with HTTP *"504 Application Error"*

```yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
[...]
specs:
- endpointSelector:
  matchLabels:
    app: myService
  ingress:
  - toPorts:
    - ports:
      - port: "8000"
        protocol: TCP
        l7proto: chaos
        l7:
        - probability: "0.5"
          rewrite-status: 504 Application Error
```

Requirements for Failure Injection

- **Proxy**
  → Envoy
- **Ability to simulate failure & probability aspect**
  → Envoy Go extension “chaos”
- **Transparency**
  → Cilium & eBPF
- **Visibility**
  → HTTP headers, Envoy metrics

Demo

Getting Started

Clone examples repo:

```
$ git clone https://github.com/cilium/chaos-testing-examples.git
```

Create Kubernetes cluster:

```
$ minikube start --network-plugin=cni --memory=4096
```

Install Cilium:

```
$ kubectl apply -f cilium-minikube.yaml
```

Deploy your app:

```
$ kubectl apply -f deathstar.yaml
$ kubectl apply -f falcon.yaml
```

Apply the fault injection policy:

```
$ kubectl apply -f examples/delay-response.yaml
```

More Examples

Delay 80% of responses by 50ms, and 20% fo responses by 1s.

(Yes, there is a probability of 0.8*0.2 to get delayed 1050ms)

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
[...]
specs:
- endpointSelector:
  matchLabels:
    app: myService
  ingress:
  - toPorts:
    - ports:
      - port: "8000"
        protocol: TCP
        l7proto: chaos
        l7:
        - probability: "0.8"
          delay-response: 50ms
        - probability: "0.2"
          delay-response: 1s
```

cilium

Rewrite all “200 OK” responses that were sent in return to a “GET” request into “404 NOT FOUND”

```yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
[...]
specs:
- endpointSelector:
  matchLabels:
    app: myService
  ingress:
  - toPorts:
    - ports:
      - port: "8000"
        protocol: TCP
        l7proto: chaos
        l7:
        - status-code: "200"
          method: GET
          rewrite-status-code: 404 NOT FOUND
```

Thank You

Join the community:

GitHub: https://github.com/cilium/cilium

Slack: https://cilium.io/slack

Twitter: @ciliumproject

All Examples:

https://github.com/cilium/chaos-testing-examples