Microservices Patterns With Envoy Sidecar Proxy: The series

I've blogged in the past about "how I'm excited for a '2.0' microservices stack" and what some of that entails. I even tried to lay out why service interaction/conversations and the network are the hardest parts of a practical microservices implementation. In this (and a next series of posts) post, I'd like to start to go a bit deeper.

With the recent announcement of the Istio Mesh project including Red Hat's support for the project, I'd like to spend the next few blogs going deep inside this technology (Envoy Proxy, Istio Mesh, etc) and explain what it does, how it works, and why IMHO it's a game changer. Follow me @christianposta to stay up with these blog post releases. I think the flow for what I cover over the next series will be something like:

- What is Envoy Proxy, how does it work?
- How to implement some of the basic patterns with Envoy Proxy?
- How Istio Mesh fits into this picture
- How Istio Mesh works, and how it enables higher-order functionality across clusters with Envoy
- How Istio Mesh auth works

In the next few blog posts specifically, I want to cover some of the client-side, service-interaction features that Envoy Proxy provides. For a quick refresher, Envoy Proxy is a small, lightweight, native/C++ application that enables the following features (and more!):

- Service discovery
- Adaptive routing / client side load balancing
- Automatic retries
- Circuit breakers
- Timeout controls
- back pressure
- Rate limiting
- Metrics/stats collection
- Tracing
- request shadowing
- Service refactoring / request shadowing
- TLS between services
- Forced service isolation / outlier detection

Blog series flow

I'm going to cover those aforemention topics in a "deep dive" approach: i.e., how each of those pieces of functionality works under specified scenarios (with demos, etc). For the first few blog posts, it will be less "how to easily use this or operationalize this" and more "what's the behavior i should come to expect at the lower levels of usage".

Here's the idea for the next couple of parts (will update the links as they're published):

- Circuit breakers (Part I)
- Retries / Timeouts (Part II)
- Distributed Tracing (Part III)
- Metrics collection with Prometheus (Part IV)
- Service Discovery (Part V)
- The next parts will cover more of the client-side functionality (Request Shadowing, TLS, etc), just not sure which parts will be which yet :)

Part II and III should be coming next week: feel free to follow along!

Demos for each topic

Each example includes full demo, configuration, helper scripts, and documentation for how to exercise the demos. The source code for these is at my github under the envoy-microservices-patterns repo. I highly recommend you take a look there.

Sidecars

Envoy is well-suited for deployment as a sidecar deployment, which means it gets deployed alongside your application (one to one) and your application interacts with the outside world through Envoy Proxy. This means, as an application developer, you can take advantage of the features provided by Envoy through configuration (like service discovery, client-side load balancing, circuit breakers, tracing, etc). Additionally, this means your

applications don't have to include lots of libraries, dependencies, transitive dependencies etc, and hope that each developer properly implements these features.

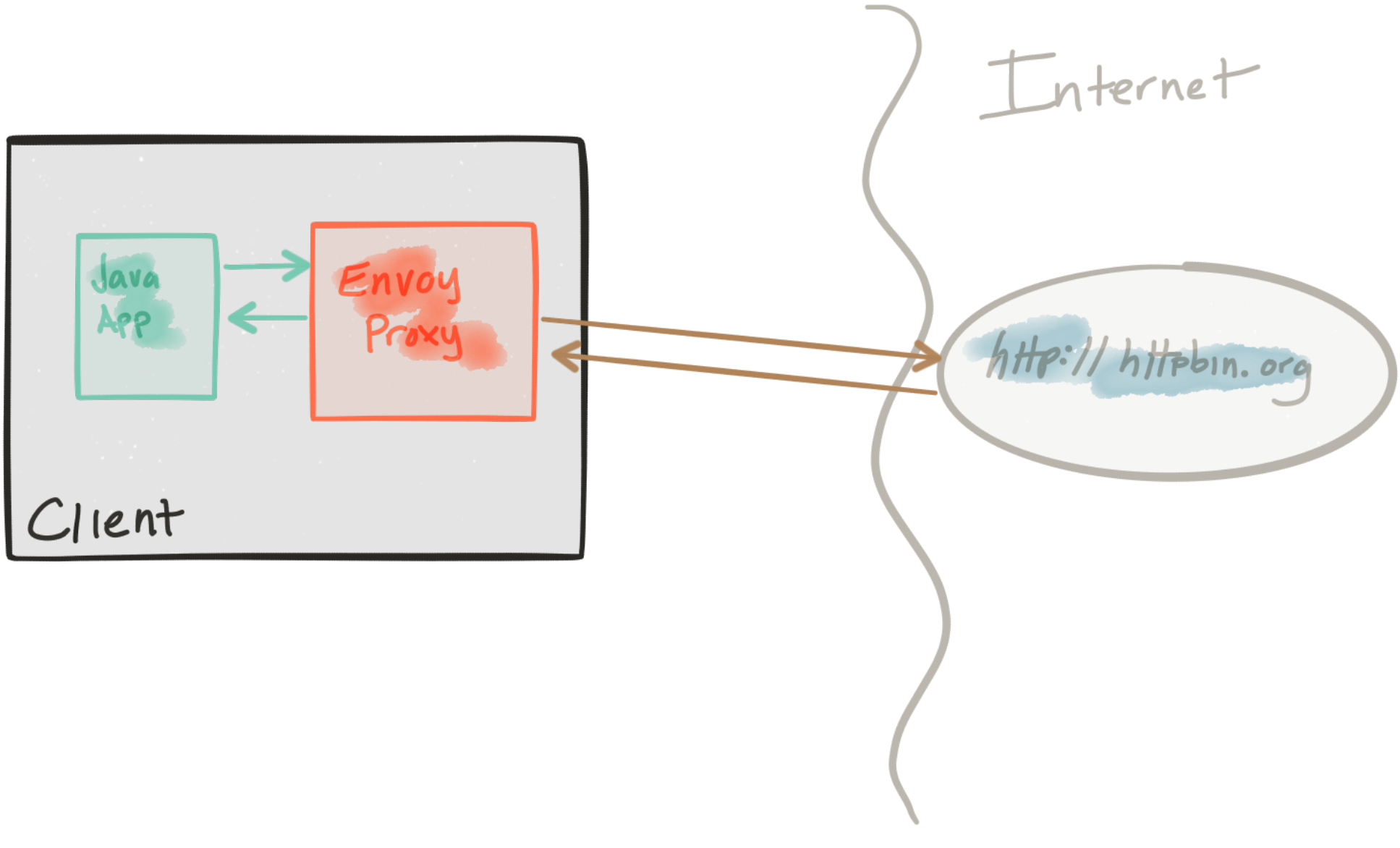
As Java developers this is great! This means we shouldn't have to include things like Netflix OSS Ribbon/Hystrix/Eureka/Tracing libraries (Ben Christensen, creator of Hystrix, gave a great talk at the Microservices Summit in 2016 explaining a bit about this…).

Overview and background

These demos are intentionally simple so that I can illustrate the patterns individually.

All of these demos are comprised of a client and a service. The client is a Java http application that simulates making http calls to the "upstream" service (note, we're using Envoys terminology here, and throught this repo). The client is packaged in a Docker image named `docker.io/ceposta/http-envoy-client:latest` . Alongside the http-client Java application is an instance of Envoy Proxy. In this deployment model, Envoy is deployed as a sidecar alongside the service (the http client in this case). When the http-client makes outbound calls (to the "upstream" service), all of the calls go through the Envoy Proxy sidecar.

The "upstream" service for these examples is httpbin.org. httpbin.org allows us to easily simulate HTTP service behavior. It's awesome, so check it out if you've not seen it.



Each demo will have it's own `envoy.json` configuration file. I definitely recommend taking a look at the reference documentation for each section of the configuration file to help understand the configuration. The good folks at datawire.io also put together a nice intro to Envoy and its configuration which you should check out too.

Please stay tuned! Part I and II are already up and more coming later next week!

SHARE ON

Microservices Patterns With Envoy Sidecar Proxy: The series was published on May 26, 2017.