Get Started with Envoy on your Laptop

A quick path to try out Envoy features in the safety of your own lap

Before running Envoy in a production setting, you might want to tour its capabilities. By us
and a few small services, you'll quickly get the idea behind using Envoy to proxy traffic, and
might have in your network architecture. We'll walk through how to run Envoy on your lap
configurations, and observe results.

```
rds.json
rds.json
2107                "header": {
2108                  "key": "x-tbn-domain",
2109                  "value": "app.turbinelabs.io:30080"
2110                }
2111              }
2112            ]
2113          },
2114          {
2115            "name": "api.turbinelabs.io-30080",
2116            "domains": [
2117              "api.turbinelabs.io"
2118            ],
2119            "routes": [
2120              {
2121                "match": {
2122                  "prefix": "/v2.0/stats/forward",
2123                  "case_sensitive": false,
2124                  "headers": [
2125                    {
2126                      "name": "X-Tbn-Stats-Api-Version",
2127                      "value": "2018-01-12-0214a"
2128                    }
2129                  ]
2130                },
2131                "route": {
Line 2146, Column 19                                  Spaces: 2
```

We'll start with somewhat fewer lines of config than this.

Requirements

While you can build Envoy from source, the easiest way to get started is by using the officia before starting out, you'll need the following software installed and configured:

- Docker
- Docker Compose
- Git
- curl

We use Docker and Docker Compose to set up and run example service topologies using En
Envoy examples, and curl to send traffic to running services.

Running Envoy

Running the latest Docker image will technically get you Envoy on your laptop, but withou
anything very interesting. Let's get a simple front proxy topology running, which will send
backends. The Envoy source repository has a couple of examples, so to start, clone that rep
`examples/front-proxy` directory. This contains Dockerfiles, config files and a Docker Comp
up a the topology.

```
$ git clone https://github.com/envoyproxy/envoy

$ cd envoy/examples/front-proxy
```

The services run a very simple Flask application, defined in `service.py`. An Envoy runs in
sidecar, configured with the `service-envoy.yaml` file. Finally, the `Dockerfile-service` cre
runs Envoy and the service on startup.

The front proxy is simpler. It runs Envoy, configured with the `front-envoy.yaml` file, and u
`frontenvoy` as its container definition.

The `docker-compose.yaml` file provides a description of how to build, package, and run the
services together.

To build our containers, run:

```
docker-compose up --build -d
```

This starts a single instance of the front proxy and two service instances, one configured as
other as "service2", `--build` means build containers before starting up, and `-d` means ru
mode.

Running `docker-compose ps` should show the following output:

```
$ docker-compose ps
          Name                        Command               State
-----------------------------------------------------------------------------------
-----------
frontproxy_front-envoy_1   /bin/sh -c /usr/local/bin/ ...   Up      0.0.0.0:8000-
0.0.0.0:8001->8001/tcp
```

```
frontproxy_service1_1      /bin/sh -c /usr/local/bin/ ...   Up      80/tcp

frontproxy_service2_1      /bin/sh -c /usr/local/bin/ ...   Up      80/tcp
```

Sending Traffic

Docker Compose has mapped port 8000 on the front-proxy to your local network. Open you http://localhost:8000/service/1, or run `curl localhost:8000/service/1` . You should see

```
$ curl localhost:8000/service/1

Hello from behind Envoy (service 1)! hostname: 6632a613837e resolvedhostname: 172
```

Going to http://localhost:8000/service/2 should result in

```
$ curl localhost:8000/service/2

Hello from behind Envoy (service 2)a! hostname: bf97b0b3294d resolvedhostname: 17
```

You’re connecting to Envoy, operating as a front proxy, which is in turn sending your reques 2.

Configuring Envoy

This is a simple way to configure Envoy statically for the purpose of demonstration. As we r
you can really harness its power by dynamically configuring it.

Let's take a look at how Envoy is configured. Inside the `docker-compose.yaml` file, you'll see
definition for the `front-envoy` service:

```
front-envoy:
    build:
      context: ../
      dockerfile: front-proxy/Dockerfile-frontenvoy
    volumes:
      - ./front-envoy.yaml:/etc/front-envoy.yaml
    networks:
      - envoymesh
    expose:
      - "80"
      - "8001"
    ports:
      - "8000:80"
      - "8001:8001"
```

Going from top to bottom, this says:

1. Build a container using the `Dockerfile-frontenvoy` file located in the current director
2. Mount the `front-envoy.yaml` file in this directory as `/etc/front-envoy.yaml`
3. Create and use a Docker network named " `envoymesh` " for this container
4. Expose ports 80 (for general traffic) and 8001 (for the admin server)
5. Map the host port 8000 to container port 80, and the host port 8001 to container port

Knowing that our front proxy uses the front-envoy.yaml file, let's take a deeper look. Our fil elements, `static_resources` and `admin` .

```
static_resources:

admin:
```

The `admin` block is relatively simple.

```
admin:

  access_log_path: "/dev/null"

  address:

    socket_address:
```

```
        address: 0.0.0.0
        port_value: 8001
```

The `access_log_path` field is set to `/dev/null`, meaning access logs to the admin server a
or production environment, users would change this value to an appropriate destination. T
Envoy to create an admin server listening on port 8001.

The `static_resources` block contains definitions for clusters and listeners that aren't dyna
cluster is a named group of hosts/ports, over which Envoy will load balance traffic, and list
network locations that clients can connect to. The `admin` block configures our admin serve

Our front proxy has a single listener, configured to listen on port 80, with a filter chain that
manage HTTP traffic.

```
listeners:
  - address:
      socket_address:
        address: 0.0.0.0
        port_value: 80
    filter_chains:
    - filters:
      - name: envoy.http_connection_manager
        config:
```

```
          codec_type: auto
          stat_prefix: ingress_http
          route_config:
            name: local_route
```

Within the configuration for our HTTP connection manager filter, there is a definition for a configured to accept traffic for all domains.

```
virtual_hosts:
            - name: backend
              domains:
              - "*"
              routes:
              - match:
                  prefix: "/service/1"
                route:
                  cluster: service1
              - match:
                  prefix: "/service/2"
                route:
                  cluster: service2
```

Routes are configured here, mapping traffic for `/service/1` and `/service/2` to the appro

Next come static cluster definitions:

```
clusters:
  - name: service1
    connect_timeout: 0.25s
    type: strict_dns
    lb_policy: round_robin
    http2_protocol_options: {}
    hosts:
    - socket_address:
        address: service1
        port_value: 80
  - name: service2
    connect_timeout: 0.25s
    type: strict_dns
    lb_policy: round_robin
    http2_protocol_options: {}
    hosts:
```

```
    - socket_address:
        address: service2
        port_value: 80
```

You can configure timeouts, circuit breakers, discovery settings, and more on clusters. Clus endpoints—a set of network locations that can serve requests for the cluster. In this examp canonically defined in DNS, which Envoy can read from. Endpoints can also be defined dire or read dynamically via the Endpoint Discovery Service.

Modifying Configuration

In Envoy, you can modify the config files, rebuild Docker images, and test the changes. Liste way of attaching additional functionality to listeners. For instance, to add access logging to the `access_log` object to your filter config, as shown here.

```
- filters:
    - name: envoy.http_connection_manager
      config:
        codec_type: auto
        stat_prefix: ingress_http
        access_log:
          - name: envoy.file_access_log
```

```
                config:
                  path: "/var/log/access.log"
            route_config:
```

Destroy your Docker Compose stack with `docker-compose down`, then rebuild it with `docke` `d`. Make a few requests to your services using curl, then log into a shell with `docker-compos` `/bin/bash`. An `access.log` file should be in `/var/log`, showing the results of your reques

Admin Server

A great feature of Envoy is the built-in admin server. If you visit `http://localhost:8001` in should see a page with links to more information. The `/clusters` endpoint shows statistic and the `stats` endpoint shows more general statistics. You can get information about the `/server_info`, and you can query and alter logging levels at `/logging`. General help is av `/help` endpoint.

Further Exploration

If you're interested in exploring more of Envoy's capabilities, the Envoy examples have mor that will get you slightly more real-world, but still use statically discovered configurations. more about how to operate Envoy in a production setting, the service discovery integration walks through what it means to integrate Envoy with your existing environment.