Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Two: API Simulation Basics

Purpose of this Lab

- Learn how to use the ‘hoverctl’ CLI to orchestrate Hoverfly
- Learn how to use Hoverfly to record a request to an API
- Learn how to use Hoverfly to simulate a request to an API
- Gain a basic understanding of a ‘simulation’

Questions

Feel free to ask questions and ask for help at any point during the workshop. Also, don’t be afraid to collaborate with others.

Pre-requisites

Make sure you have cloned the api-simulation-training Git repository, and that you are in the correct directory!

```
$ cd api-simulation-training/2-api-simulation-basics
```

You can also enabled Bash autocompletion for Hoverctl by execution the following command

```
$ hoverctl completion
```

Exercise: Capture and Simulating a Request with Hoverfly

During this exercise, you will become familiar with orchestrating Hoverfly using hoverctl, and will then use it to capture and simulate a request to an API. In this case, there is no problem to solve - you must simply follow each step exactly in order to complete the exercise.

Steps

1. First of all, start an instance of Hoverfly:

```
$ hoverctl start
Hoverfly is now running

+------------+------+
| admin-port | 8888 |
| proxy-port | 8500 |
+------------+------+
```

At any time during the tutorials you can check whether Hoverfly is running, which ports it is listening on, by issuing the hoverctl status command:

```
$ hoverctl status

+------------+----------+
| Hoverfly   | running  |
| Admin port |     8888 |
| Proxy port |     8500 |
| Mode       | capture  |
| Middleware | disabled |
+------------+----------+
```

2. Start the flights service, and make a request to it to validate that it is running. Here we are just searching for all the flights that are available tomorrow (note that your results from the curl may appear different than the results shown here, as the flights service is non-deterministic!):

```
$ ./run-flights-service.sh
waiting for service to start
waiting for service to start
waiting for service to start
service started

$ curl localhost:8081/api/v1/flights?plusDays=1 | jq
[
  {
    "origin": "Berlin",
    "destination": "New York",
    "cost": "617.31",
    "when": "03:45"
  },
  {
    "origin": "Amsterdam",
    "destination": "Dubai",
    "cost": "3895.49",
    "when": "21:20"
  },
  {
    "origin": "Milan",
    "destination": "New York",
    "cost": "4950.31",
    "when": "08:49"
  }
]
```

Tip: We are piping our output into jq in order to format the JSON in a way which is human readable.

3. Now move Hoverfly into 'capture' mode. During this mode, any request that is intercepted by Hoverfly will be captured:

```
$ hoverctl mode capture
Hoverfly has been set to capture mode
```

Tip: Remember, if we ever forget then we can do `hoverctl --help` to get the list of available commands.

4. Once in capture mode, make a request to the flights API, but this time specify Hoverfly as a proxy (note that your curled flight results may look different than that shown below):

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "20:53"
  },
  {
    "origin": "Amsterdam",
    "destination": "Boston",
    "cost": "2999.69",
    "when": "19:45"
  }
]
```

By specifying the proxy, it means the request will first go to the proxy (Hoverfly), and then be forwarded onto the real flights API afterwards. This reverse is true for the response, and this is how Hoverfly is able intercept network traffic.

5. Look at the logs to make sure Hoverfly has intercepted the request:

```
$ hoverctl logs
INFO[2017-09-20T11:38:48+01:00] Mode has been changed
mode=capture
INFO[2017-09-20T11:40:28+01:00] request and response captured
mode=capture request=&map[headers:map[Accept:[*/*]
Proxy-Connection:[Keep-Alive] User-Agent:[curl/7.54.0]] body: method:GET
scheme:http destination:localhost:8081 path:/api/v1/flights
query:map[plusDays:[1]]] response=&map[error:nil response]
...
```

Tip: You can also follow the logs with the follow argument: `hoverctl logs --follow`

6. Now, let's take a look at the simulation that we have produced, by exporting it and then opening it in a text editor (we use atom, but feel free to substitute your favourite like vim or emacs in the command below):

```
$ hoverctl export module-two-simulation.json
Successfully exported simulation to module-two-simulation.json
$ atom module-two-simulation.json
```

Take a look at the simulation file, and see if you recognise your recorded data. The request you captured should correspond to a single element in the pairs array.

7. Now, we can use our simulation to simulate the flights API. First, stop the flights service to make we are unable to communicate with it, and then curl it (**without** a proxy argument) to verify it's happened:

```
$ ./stop-flights-service.sh
service successfully shut down
$ curl localhost:8081/api/v1/flights?plusDays=1
curl: (7) Failed to connect to localhost port 8081: Connection refused
```

Tip: Note that when we are expecting a curl to fail (or we've already seen it fail once) then **execute the curl without piping the results to jq**. Otherwise we are piping the error in non-JSON format to jq, and it will get confused!

8. Now, put Hoverfly into simulate mode:

```
$ hoverctl mode simulate
Hoverfly has been set to simulate mode with a matching strategy of
'strongest'
```

During simulate mode, instead of forwarding on the traffic to the real API, Hoverfly will immediately respond to the client with our recorded request.

9. Now we can repeat our request, only this time using Hoverfly as a proxy. We should now receive our recorded response rather than an error:

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "20:53"
  },
  {
    "origin": "Amsterdam",
    "destination": "Boston",
    "cost": "2999.69",
    "when": "19:45"
  }
]
```

We have successfully simulated our first API endpoint!