# Effective Testing with API Simulation and (Micro)Service Virtualisation

# Module One: 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 we are in the correct directory!

```
$ cd ../1-api-simulation-basics
```

## 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 a 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 |
+------------+------+
```

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:

```
$ ./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"
  }
]
```

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:

```
$ 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
...
```

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:

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

```
$ ./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
```

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"
  }
]
```