Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Six: Fault Injection Testing

Purpose of this Lab

- Understanding the concept of fault injection
- Use Hoverfly to add delay to recorded simulation responses
- Use Hoverfly to modify responses with the goal of injecting faults

Questions

Feel free to ask questions and ask for help at any point during the workshop. Also, please 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/6-fault-injection
```

Exercise 1: Adding Delays to a Response

During this exercise, we will learn:

- How to add delays to recorded simulation responses. This approach to fault injection can be used within a test suite to confirm a consumer/client can handle a delayed response from a supplier
- How to target specific simulation URLs to add delay to

Steps

1. Make sure Hoverfly is running:

```
$ hoverctl start
target Hoverfly is already running
```

2. Make sure the flights API is running:

```
$ ./run-flights-service.sh
service started
```

4. Make sure all the existing data is deleted from Hoverfly:

```
$ hoverctl delete
Are you sure you want to delete the current simulation? [y/n]: y
Simulation data has been deleted from Hoverfly

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

5. Now move Hoverfly into ‘capture’ mode.

```
$ 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

6. Once in capture mode, make a request to the flights API, making sure to 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"
  }
]
```

Make an additional request to the discount-codes endpoint on the same service

```
$ curl localhost:8081/api/v1/discount-codes --proxy localhost:8500 | jq
[
  "SALE10",
  "SALE20",
  "SPECTO"
]
```

6. Export the simulation 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-six-simulation.json
Successfully exported simulation to module-six-simulation.json
$ atom module-six-simulation.json
```

Locate the “globalActions” property, and add a global delay by changing the property contents to the following:

```
"globalActions": {
  "delays": [
    {
      "urlPattern": ".",
      "httpMethod": "",
      "delay": 2000
    }
  ]
}
```

Save the file and import this into Hoverfly. Stop the flights API, and change the Hoverfly mode to simulate. Make the same request to the flights API as you did earlier:

```
$ ./stop-flights-service.sh
service successfully shut down
$ hoverctl mode simulate
Hoverfly has been set to simulate mode with a matching strategy of
'strongest'
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "20:53"
  },
  ...
]
```

Did you notice a delay in response? If not, try again but use the Linux ‘time’ command as a prefix to the curl in order to record and display the time taken to execute the command:

```
$ time curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500
| jq
[
  {
    "origin": "Milan",
    "destination": "Boston",
    "cost": "879.31",
    "when": "23:03"
  },
  ...
]

real  0m2.021s
user  0m0.007s
sys   0m0.006s
```

Notice that the “real” time is shown to be greater than 2 seconds (you can remove the delay in the simulation data, re-import and issue a time curl command again to observe what is shown without the delay).

Advanced Exercise

Assume that our tests only want to add a delay to the flight endpoint. What happens if we attempt to curl the discount-codes with our current delay configuration?

```
$ time curl localhost:8081/api/v1/discount-codes --proxy localhost:8500 |
jq
[
  "SALE10",
  "SALE20",
  "SPECTO"
]

real  0m2.018s
user  0m0.010s
sys   0m0.007s
```

The delay is also applied to this request. Have a look at the Hoverfly Delay documentation and see if you can configure the delay to only be applied to the flights request.

Exercise 2: Modifying the Response with Middleware

During this exercise, we will learn:

- How to use middleware to intercept and modify the request/response made to a simulated endpoint in Hoverfly
- How to write middleware in Python to modify the response HTTP status code and data
- How to write middleware in JavaScript to modify the response HTTP status code and data

Steps

1. Make sure Hoverfly is running:

```
$ hoverctl start
target Hoverfly is already running
```

2. Make sure the flights API is running:

```
$ ./run-flights-service.sh
service started
```

3. Make sure all the existing data is deleted from Hoverfly:

```
$ hoverctl delete
Are you sure you want to delete the current simulation? [y/n]: y
Simulation data has been deleted from Hoverfly
```

4. Now move Hoverfly into 'capture' mode.

```
$ 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

5. Once in capture mode, make a request to the flights API, making sure to 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"
  }
]
```

6. Stop the flight API and switch Hoverfly in to simulate mode:

```
$ ./stop-flights-service.sh
```

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

You can now curl the flight API endpoint with Hoverfly as a proxy in order to see the captured data returned.

7. Now we will introduce middleware into Hoverfly. This step assumes that you have installed Python on your local machine, as Hoverfly will run the python binary in order to execute the Python middleware.

Create a file named middleware.py in the current directory with the following content:

```
#!/usr/bin/env python

import sys
import json
import logging
import random

logging.basicConfig(filename='middleware.log', level=logging.DEBUG)
logging.debug('Middleware "modify_request" called')

def main():
    payload = sys.stdin.readlines()[0]

    logging.debug(payload)

    payload_dict = json.loads(payload)
    payload_dict['response']['status'] = 200

    if "response" in payload_dict and "body" in payload_dict["response"]:
        payload_dict["response"]["body"] = "[{\"origin\":
\"Mars\",\"destination\": \"Pluto\",\"cost\": \"9999.99\",\"when\":
\"00:00\"}\n]"

    print(json.dumps(payload_dict))

if __name__ == "__main__":
    main()
```

Even if you are not a Python programmer, have a look through the code. Can you understand what is happening?

8. Configure Hoverfly to use the Python middleware we have just created:

```
$ hoverctl middleware --binary python --script middleware.py
Hoverfly middleware configuration has been set to
Binary: python
Script: #!/usr/bin/env python

import sys
import json
import logging
...
```

Curl the simulated flights API, and observe the data returned (this is not what we captured!):

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Mars",
    "destination": "Pluto",
    "cost": "9999.99",
    "when": "00:00"
  }
]
```

9. See if you can modify the Python script to return a 401 HTTP Status code, or an error response of your choosing. Don't forget to load any modifications to the middleware into Hoverfly each time you make a change (Hoverfly does not dynamically reload the script contents):

```
$ hoverctl middleware --binary python --script middleware.py
Hoverfly middleware configuration has been set to
Binary: python
Script: #!/usr/bin/env python

import sys
import json
```

```
import logging
...
```

Think about the types of deterministic error conditions you could add to the simulation when testing a client calling the simulated API!

10. If you have node.js installed on your local machine you can try and run the following JavaScript middleware to perform similar functionality as the Python version above:

```
#!/usr/bin/env node

process.stdin.resume();
process.stdin.setEncoding('utf8');
process.stdin.on('data', function(data) {
  var parsed_json = JSON.parse(data);
  parsed_json.response.status = 200;
  parsed_json.response.body = JSON.stringify([
    {
      "origin": "Mars",
      "destination": "Pluto",
      "cost": "9999.99",
      "when": "00:00"
    }
  ]);

  process.stdout.write(JSON.stringify(parsed_json));
});
```

Advanced Exercise

Create some middleware in the language of your choice that does the following:

1. Simulate the flights API endpoint, and append the string " (Coach Class)" to all the destination fields returned e.g. "destination": "London (Coach Class)"
2. Simulate the flights API endpoint and returns a 404 HTTP status code **only** if the from or to field is set to "Londonium" and the plus days value is greater than 14

3. Simulate the flights API endpoint, and return a 503 HTTP status code on every fifth execution of the flights endpoint (Hint: Hoverfly creates a new process to run the middleware every time the middleware is triggered, and so you will need to manage your middleware's state accordingly)