Hoverfly Documentation

Release v1.9.1

SpectoLabs

Apr 27, 2024

CONTENTS

Hoverfly is a lightweight, open source API simulation tool. Using Hoverfly, you can create realistic simulations of the APIs your application depends on.

- Replace slow, flaky API dependencies with realistic, re-usable simulations

- Simulate network latency, random failures or rate limits to test edge-cases

- Extend and customize with any programming language

- Export, share, edit and import API simulations

- CLI and native language bindings for Java and Python

- REST API

- Lightweight, high-performance, run anywhere

- Apache 2 license

SOURCE

The Hoverfly source code is available on GitHub.

Hoverfly is developed and maintained by SpectoLabs.

TWO

CONTENTS

2.1 Introduction

2.1.1 Motivation

Developing and testing interdependent applications is difficult. Maybe you're working on a mobile application that needs to talk to a legacy API. Or a microservice that relies on two other services that are still in development.

The problem is the same: how do you develop and test against external dependencies which you cannot control?

You could use mocking libraries as substitutes for external dependencies. But mocks are intrusive, and do not allow you to test all the way to the architectural boundary of your application.

Stubbed services are better, but they often involve too much configuration or may not be transparent to your application.

Then there is the problem of managing test data. Often, to write proper tests, you need fine-grained control over the data in your mocks or stubs. Managing test data across large projects with multiple teams introduces bottlenecks that impact delivery times.

Integration testing "over the wire" is problematic too. When stubs or mocks are swapped out for real services (in a continuous integration environment for example) new variables are introduced. Network latency and random outages can cause integration tests to fail unexpectedly.

Hoverfly was designed to provide you with the means to create your own "dependency sandbox": a simulated development and test environment that you control.

Hoverfly grew out of an effort to build "the smallest service virtualization tool possible".

2.1.2 Download and installation

Hoverfly comes with a command line interface called **hoverctl**. Archives containing the Hoverfly and hoverctl binaries are available for the major operating systems and architectures.

- MacOS 64bit
- Linux 32bit
- Linux 64bit
- Windows 32bit
- Windows 64bit

Download the correct archive, extract the binaries and place them in a directory on your PATH.

Homebrew (MacOS)

If you have homebrew, you can install Hoverfly using the `brew` command.

```
brew install SpectoLabs/tap/hoverfly
```

To upgrade your existing hoverfly to the latest release:

```
brew upgrade hoverfly
```

To show which versions are installed in your machine:

```
brew list --versions hoverfly
```

You can switch to a previously installed version as well:

```
brew switch hoverfly <version>
```

To remove old versions :

```
brew cleanup hoverfly
```

Docker

If you have Docker, you can run Hoverfly using the `docker` command.

```
docker run -d -p 8888:8888 -p 8500:8500 spectolabs/hoverfly:latest
```

This will run the latest version of the Hoverfly Docker image.

You can also pass Hoverfly configuration flags when starting with Docker. For example if you need to run Hoverfly in webserver mode:

```
docker run -d -p 8888:8888 -p 8500:8500 spectolabs/hoverfly:latest -webserver
```

This Docker image does not contain hoverctl. Our recommendation is to have hoverctl on your host machine and then configure hoverctl to use the newly started Hoverfly Docker instance as a new target.

See also:

For a tutorial of creating a new target in hoverctl, see *Controlling a remote Hoverfly instance with hoverctl*.

Kubernetes

You can use Helm to install Hoverfly directly to your Kubernetes cluster. Hoverfly chart is available from the official Helm incubator repo.

Use `helm repo add` to add the incubator repo if you haven't done so:

```
helm repo add incubator https://kubernetes-charts-incubator.storage.googleapis.com/
```

Here is the command for a basic Hoverfly installation with the default settings:

```
helm install incubator/hoverfly
```

The default installation create a `ClusterIP` type service, which makes it only reachable from within the cluster.

After the installation, you can use port forwarding to access the application on localhost:

```
kubectl port-forward $HOVERFLY_POD_NAME 8888 8500
```

You can find the `HOVERFLY_POD_NAME` by doing `kubectl get pod`

See more details on Helm Charts project.

2.1.3 Getting Started

> **Note**
>
> It is recommended that you keep Hoverfly and hoverctl in the same directory. However if they are not in the same directory, hoverctl will look in the current directory for Hoverfly, then in other directories on the PATH.

Hoverfly is composed of two binaries: **Hoverfly** and **hoverctl**.

hoverctl is a command line tool that can be used to configure and control Hoverfly. It allows you to run Hoverfly as a daemon.

Hoverfly is the application that does the bulk of the work. It provides the proxy server or webserver, and the API endpoints.

Once you have extracted both Hoverfly and hoverctl into a directory on your PATH, you can run hoverctl and Hoverfly.

```
hoverctl version
hoverfly -version
```

Both of these commands should return a version number. Now you can run an instance of Hoverfly:

```
hoverctl start
```

Check whether Hoverfly is running with the following command:

```
hoverctl logs
```

The logs should contain the string `serving proxy`. This indicates that Hoverfly is running.
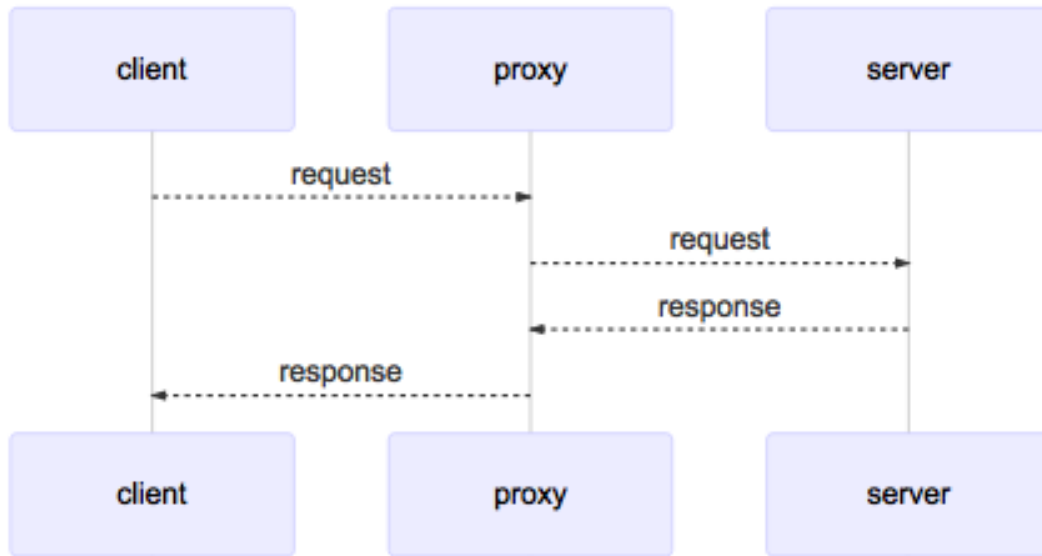
Finally, stop Hoverfly with:

```
hoverctl stop
```

2.2 Key Concepts

Hoverfly's functionality is quite broad. You are encouraged to take the time to understand these key concepts before jumping into the *Tutorials*.

2.2.1 Hoverfly as a proxy server

A proxy server passes requests between a client and server.

It is sometimes necessary to use a proxy server to reach a network (as a security measure, for example). Because of this, all network-enabled software can be configured to use a proxy server.

The relationship between clients and servers via a proxy server can be one-to-one, one-to-many, many-to-one, or many-to-many.

By default Hoverfly starts as a proxy server.

Using a proxy server

Applications can usually be configured to use a proxy server by setting environment variables:

```
export HTTP_PROXY="http://proxy-address:port"
export HTTPS_PROXY="https://proxy-address:port"
```

Launching network-enabled software within an environment containing these variables *should* make the application use the specified proxy server. The term *should* is used as not all software respects these environment variables for security reasons.

Alternatively, applications themselves can usually be configured to use a proxy. Curl can be configured to use a proxy via flags.

```
curl http://hoverfly.io --proxy http://proxy-ip:port
```

Note: The proxy configuration methods described here are intended to help you use the code examples in this documentation. The method of configuring an application or operating system to use a proxy varies depending on the environment.

- Windows Proxy Settings Explained
- Firefox Proxy Settings
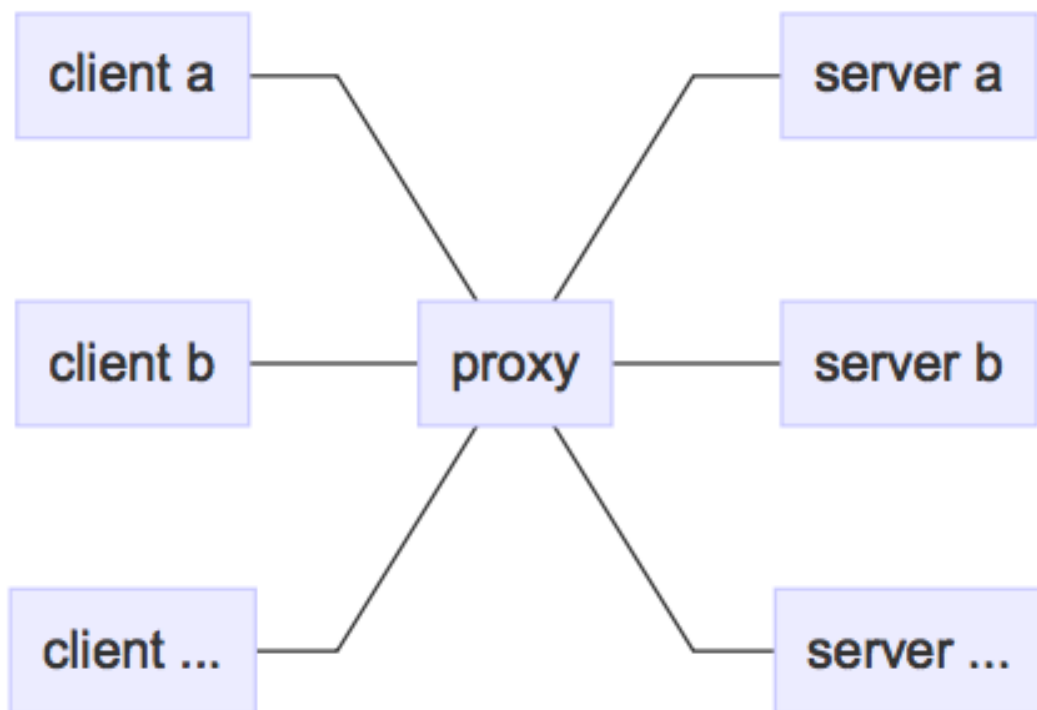
The difference between a proxy server and a webserver

A proxy server is a type of webserver. The main difference is that when a webserver receives a request from a client, it is expected to respond with whatever the intended response is (an HTML page, for example). The data it responds with is generally expected to reside on that server, or within the same network.

A proxy server is expected to pass the incoming request on to another server (the "destination"). It is also expected to set some appropriate headers along the way, such as X-Forwarded-For, X-Real-IP, X-Forwarded-Proto etc. Once the proxy server receives a response from the destination, it is expected to pass it back to the client.

2.2.2 Hoverfly as a webserver

Sometimes you may not be able to configure your client to use a proxy, or you may want to explicitly point your application at Hoverfly. For this reason, Hoverfly can run as a webserver.
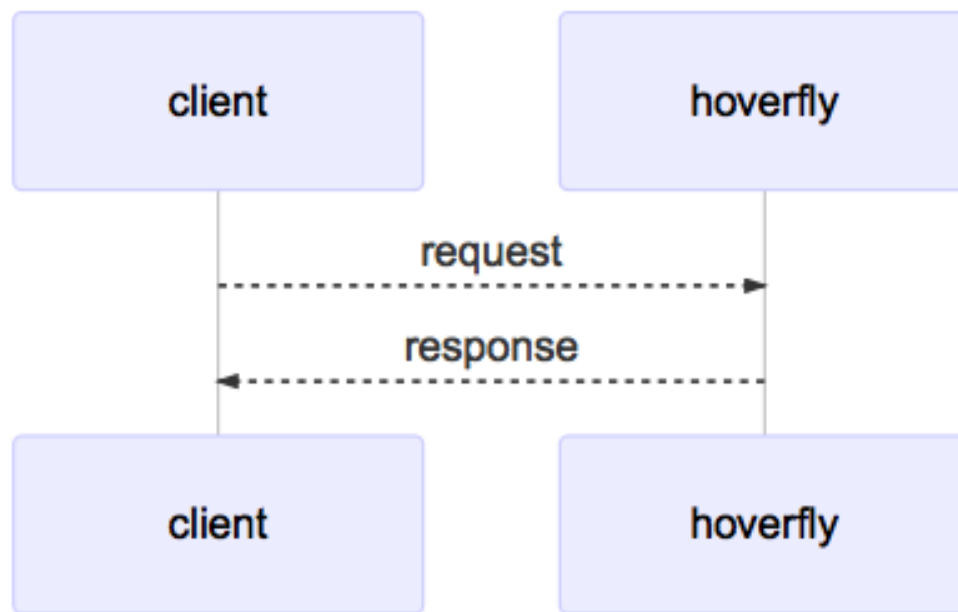
Note: When running as a webserver, Hoverfly cannot capture traffic (see *Capture mode*) - it can only be used to simulate and synthesize APIs (see *Simulate mode* and *Synthesize mode*). For this reason, when you use Hoverfly as a webserver, you should have Hoverfly simulations ready to be loaded.

When running as a webserver, Hoverfly strips the domain from the endpoint URL. For example, if you made requests to the following URL while capturing traffic with Hoverfly running as a proxy:

```
http://echo.jsontest.com/key/value
```

And Hoverfly is running in simulate mode as a webserver on:

```
http://localhost:8888
```

Then the URL you would use to retrieve the data from Hoverfly would be:

```
http://localhost:8500/key/value
```

See also:

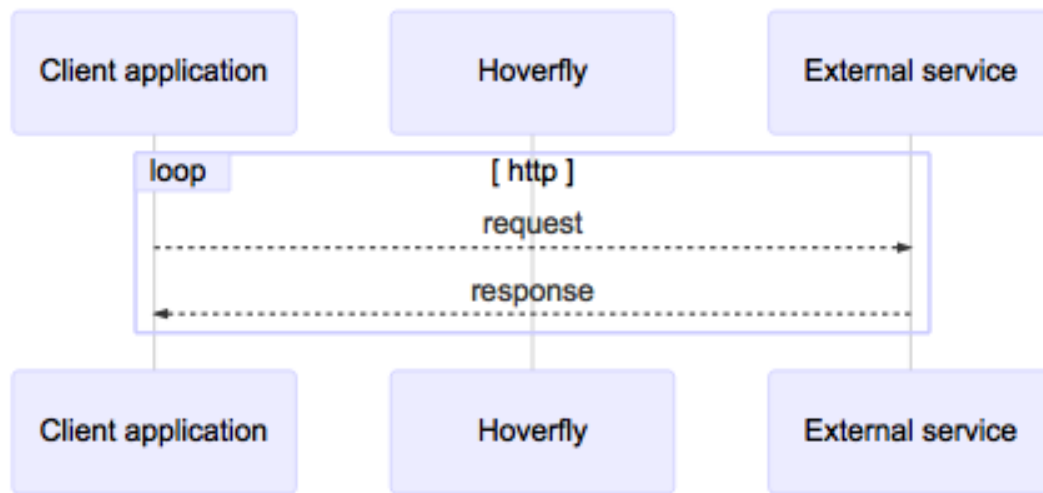Please refer to the *Running Hoverfly as a webserver* tutorial for a step-by-step example.

2.2.3 Hoverfly modes

Hoverfly has six different modes. It can only run in one mode at any one time.

Capture mode

Capture mode is used for creating API simulations.

Note: Hoverfly cannot be set to Capture mode when running as a webserver (see *Hoverfly as a webserver*).

In Capture mode, Hoverfly (running as a proxy server - see *Hoverfly as a proxy server*) intercepts communication between the client application and the external service. It transparently records outgoing requests from the client and the incoming responses from the service API.

Most commonly, requests to the external service API are triggered by running automated tests against the application that consumes the API. During subsequent test runs, Hoverfly can be set to run in *Simulate mode*, removing the dependency on the real external service API. Alternatively, requests can be generated using a manual process.

Usually, Capture mode is used as the starting point in the process of creating an API simulation. Captured data is then exported and modified before being re-imported into Hoverfly for use as a simulation.

By default, Hoverfly will ignore duplicate requests if the request has not changed. This can be a problem when trying to capture a stateful endpoint that may return a different response each time you make a request.

Using the stateful mode argument when setting Hoverfly to capture mode will disable the duplicate request check,
> enabling you to capture sequences of responses and play them back in *Simulate mode* in order.

See also:

This functionality is best understood via a practical example: see *Capturing a stateful sequence of responses* in the *Tutorials* section.

Simulate mode

In this mode, Hoverfly uses its simulation data in order to simulate external APIs. Each time Hoverfly receives a request, rather than forwarding it on to the real API, it will respond instead. No network traffic will ever reach the real external API.
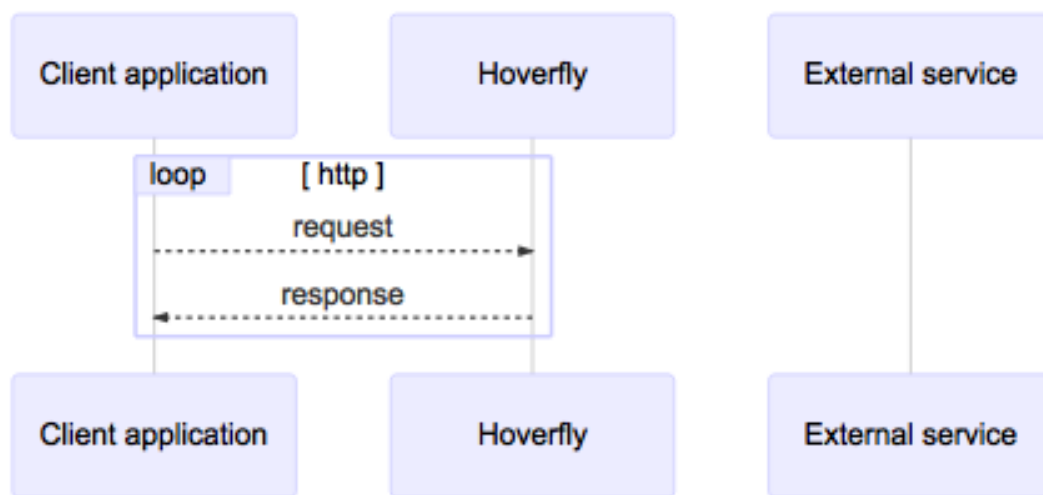
The simulation can be produced automatically via by running Hoverfly in *Capture mode*, or created manually. See *Simulations* for information.

Spy mode

In this mode, Hoverfly simulates external APIs if a request match is found in simulation data (See *Simulate mode*), otherwise, the request will be passed through to the real API.

Synthesize mode

This mode is similar to *Simulate mode*, but instead of looking for a response in stored simulation data, the request is passed directly to a user-supplied executable file. These files are known as *Middleware*.

In Synthesize mode, the middleware executable is expected to generate a response to the incoming request "on the fly". Hoverfly will then return the generated response to the application. For Hoverfly to operate in Synthesize mode, a middleware executable must be specified.

Note: You might use this mode to simulate an API that may be too difficult to record correctly via *Capture mode*. An example would be an API that uses state to change the responses. You could create middleware that manages this state

---
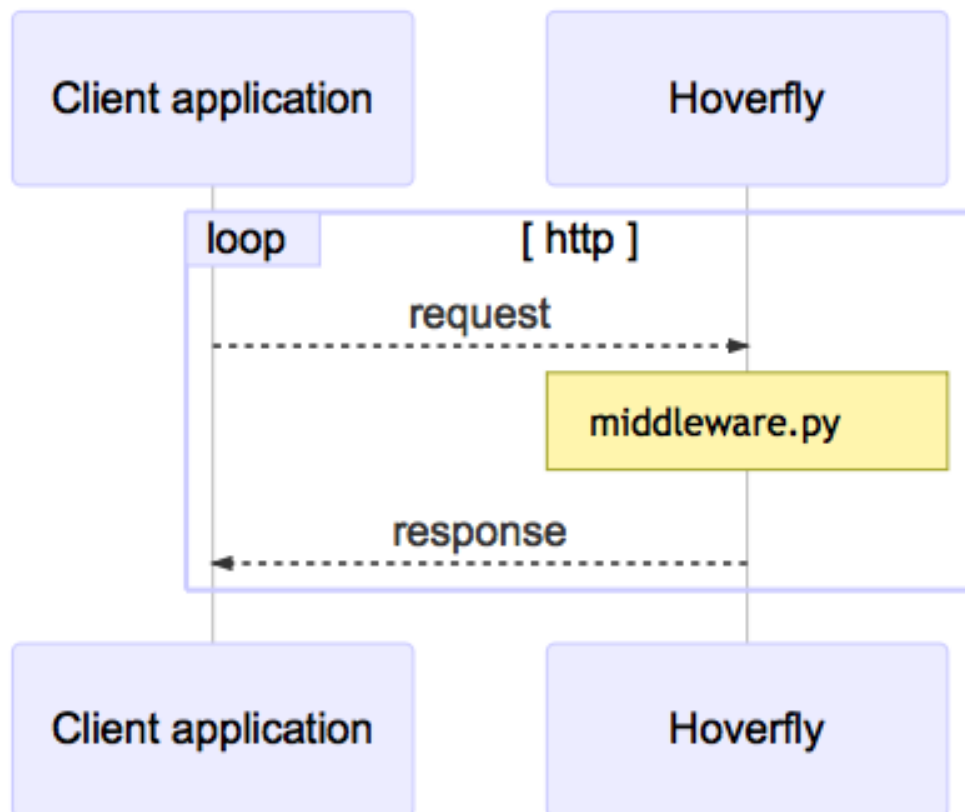
and produces the desired response based on the data in the request.

Modify mode

Note: Hoverfly cannot be set to Modify mode when running as a webserver (see *Hoverfly as a webserver*).

Modify mode is similar to *Capture mode*, except it **does not save the requests and responses**. In Modify mode, Hoverfly will pass each request to a *Middleware* executable before forwarding it to the destination. Responses will also be passed to middleware before being returned to the client.
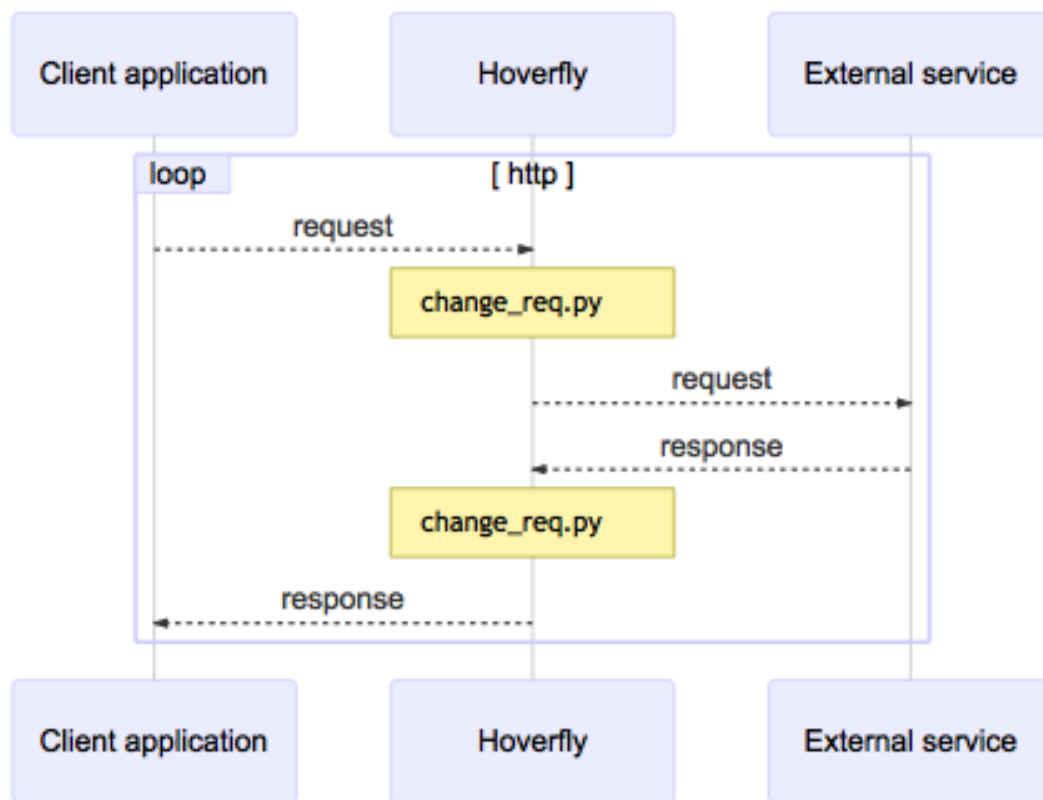
You could use this mode to "man in the middle" your own requests and responses. For example, you could change the API key you are using to authenticate against a third-party API.

Diff mode

In this mode, Hoverfly forwards a request to an external service and compares a response with currently stored simulation. With both the stored simulation response and the real response from the external service, Hoverfly is able to detect differences between the two. When Hoverfly has finished comparing the two responses, the difference is stored and the incoming request is served the real response from the external service.

The differences can be retrieved from Hoverfly using the API (*GET /api/v2/diff*). The response contains a list of differences, containing the request and the differences of the response.

```
{
  "diff": [{
    "request": {
      "method": "GET",
      "host": "time.jsontest.com",
      "path": "/",
      "query": ""
    },
    "diffReports": [{
      "timestamp": "2018-03-16T17:45:40Z",
      "diffEntries": [{
        "field": "header/X-Cloud-Trace-Context",
        "expected": "[ec6c455330b682c3038ba365ade6652a]",
        "actual": "[043c9bb2eafa1974bc09af654ef15dc3]"
      }, {
        "field": "header/Date",
        "expected": "[Fri, 16 Mar 2018 17:45:34 GMT]",
        "actual": "[Fri, 16 Mar 2018 17:45:41 GMT]"
      }, {
        "field": "body/time",
        "expected": "05:45:34 PM",
        "actual": "05:45:41 PM"
      }, {
        "field": "body/milliseconds_since_epoch",
        "expected": "1.521222334104e+12",
        "actual": "1.521222341017e+12"
      }]
    }]
  }]
}
```

This data is stored and kept until the Hoverfly instance is stopped or the the storage is cleaned by calling the API (*DELETE /api/v2/diff*).

See also:

For more information on the API to retrieve differences, see *REST API*.

2.2.4 Simulations

The core functionality of Hoverfly is to capture HTTP(S) traffic to create API simulations which can be used in testing. Hoverfly stores captured traffic as *simulations*.

Simulation JSON can be exported, edited and imported in and out of Hoverfly, and can be shared among Hoverfly users or instances. Simulation JSON files must adhere to the Hoverfly *Simulation schema*.

Simulations consist of **Request Matchers and Responses**, **Delays** and **Metadata** ("Meta").

Request Responses Pairs

Hoverfly simulates APIs by matching **incoming requests** from the client to **stored requests**. Stored requests have an associated **stored response** which is returned to the client if the match is successful.

The matching logic that Hoverfly uses to compare incoming requests with stored requests can be configured using **Request Matchers**.

Request Matchers

When Hoverfly captures a request, it creates a Request Matcher for each field in the request. A Request Matcher consists of the request field name, the type of match which will be used to compare the field in the incoming request to the field in the stored request, and the request field value.

By default, Hoverfly will set the type of match to `exact` for each field.

See also:

There are many types of Request Matcher. Please refer to *Request matchers* for a list of the types available, and examples of how to use them.

There also are two different matching strategies: **strongest match** (default) and **first match** (legacy). Please refer to *Matching strategies* for more information.

An example Request Matcher Set might look like this:

Field	Matcher Type	Value
scheme	exact	"https"
method	exact	"GET"
destination	exact	"docs.hoverfly.io"
path	exact	"/pages/keyconcepts/templates.html"
query	exact	"query=true"
body	exact	""
headers	exact	

In the Hoverfly simulation JSON file, this Request Matcher Set would be represented like this:

```
1       "request": {
2           "path": [
3               {
4                   "matcher": "exact",
5                   "value": "/pages/keyconcepts/templates.html"
6               }
7           ],
8           "method": [
```

(continues on next page)

```
 9                    {
10                        "matcher": "exact",
11                        "value": "GET"
12                    }
13                ],
14                "destination": [
15                    {
16                        "matcher": "exact",
17                        "value": "docs.hoverfly.io"
18                    }
19                ],
20                "scheme": [
21                    {
22                        "matcher": "exact",
23                        "value": "http"
24                    }
25                ],
26                "body": [
27                    {
28                        "matcher": "exact",
29                        "value": ""
30                    }
31                ],
32                "query": {
33                  "query": [
34                      {
35                          "matcher": "exact",
36                          "value": "true"
37                      }
38                  ]
39                }
40            },
```

View entire simulation file

The matching logic that Hoverfly uses to compare an incoming request to a stored request can be changed by editing the Request Matchers in the simulation JSON file.

It is not necessary to have a Request Matcher for every request field. By omitting Request Matchers, it is possible to implement **partial matching** - meaning that Hoverfly will return one stored response for multiple incoming requests.

For example, this Request Matcher will match any incoming request to the `docs.hoverfly.io` destination:

```
1                "destination": [
2                    {
3                        "matcher": "exact",
4                        "value": "docs.hoverfly.io"
5                    }
6                ]
```

View entire simulation file

In the example below, the `globMatch` Request Matcher type is used to match any subdomain of `hoverfly.io`:

```
1            "destination": [
2                {
3                    "matcher": "glob",
4                    "value": "*.hoverfly.io"
5                }
6            ]
```

View entire simulation file

It is also possible to use more than one Request Matcher for each field.

In the example below, a `regexMatch` **and** a `globMatch` are used on the `destination` field.

This will match on any subdomain of `hoverfly.io` which begins with the letter d. This means that incoming requests to `docs.hoverfly.io` and `dogs.hoverfly.io` will be matched, but requests to `cats.hoverfly.io` will not be matched.

```
1            "destination": [
2                {
3                    "matcher": "glob",
4                    "value": "*.hoverfly.io"
5                },
6                {
7                    "matcher": "regex",
8                    "value": "(\\Ad)"
9                }
10            ]
```

View entire simulation file

See also:

There are many types of Request Matcher. Please refer to *Request matchers* for a list of the types available, and examples of how to use them.

For a practical example of how to use a Request Matcher, please refer to *Loose request matching using a Request Matcher* in the tutorials section.

There alse are two different matching strategies: **strongest match** (default) and **first match** (legacy). Please refer to *Matching strategies* for more information.

Responses

Each Request Matcher Set has a response associated with is. If the request match is successful, Hoverfly will return the response to the client.

```
1            "response": {
2                "status": 200,
3                "body": "Response from docs.hoverfly.io/pages/keyconcepts/templates.html",
4                "encodedBody": false,
5                "headers": {
6                    "Hoverfly": [
7                        "Was-Here"
8                    ]
9                },
```

```
10              "templated": false
11          }
```

View entire simulation file

Editing the fields in response, combined with editing the Request Matcher set, makes it possible to configure complex request/response logic.

Binary data in responses

JSON is a text-based file format so it has no intrinsic support for binary data. Therefore if Hoverfly a response body contains binary data (images, gzipped, etc), the response body will be base64 encoded and the *encodedBody* field set to true.

```
1              "body": "YmFzZTY0IGVuY29kZWQ=",
2              "encodedBody": true,
```

View entire simulation file

Serving response bodies from files

Starting from version **1.3.0** hoverfly can return response body from a specific file:

```
"response": {
  "status": 200,
  "encodedBody": false,
  "templated": false,
  "bodyFile": "responses/200-success.json"
}
```

response/200-success.json is resolved against the directory specified in `-response-body-files-path` which is your current working directory by default.

When both `body` and `bodyFile` are specified, `body` takes precedence.

`bodyFile` is read into memory only on simulation import, not in runtime.

Reading response bodies sometimes might not be comfortable. Imagine a developer team that needs a single updating "files provider". Syncing to this provider can be a challenge and that's why hoverfly supports downloading response bodies from external urls:

```
"response": {
  "status": 200,
  "encodedBody": false,
  "templated": false,
  "bodyFile": "https://raw.githubusercontent.com/SpectoLabs/hoverfly/master/core/
→handlers/v2/schema.json"
}
```

Like local files, this feature is supported only on simulation import. To escape security issues there's another mandatory option to specify `-response-body-files-allow-origin` that lets you explicitly set the collections of urls to allow body files to be downloaded from. For the above to work you need to run hoverfly like this:

```
hoverfly -response-body-files-allow-origin="https://raw.githubusercontent.com/"
```

Delays

Once you have created a simulated service by capturing traffic between your application and an external service, you may wish to make the simulation more "realistic" by applying latency to the responses returned by Hoverfly.

Hoverfly can be configured to apply delays to responses based on URL pattern matching or HTTP method. This is done using a regular expression to match against the URL, a delay value in milliseconds, and an optional HTTP method value.

See also:

This functionality is best understood via a practical example: see *Adding delays to a simulation* in the *Tutorials* section.

You can use *Log-normal distributions of delay* for realistic delay.

You can also apply delays to simulations using *Middleware* (see the *Using middleware to simulate network latency* tutorial). Using middleware to apply delays sacrifices performance for flexibility.

Meta

The last part of the simulation schema is the meta object. Its purpose is to store metadata that is relevant to your simulation. This includes the simulation schema version, the version of Hoverfly used to export the simulation and the date and time at which the simulation was exported.

```
1      },
2    "meta": {
3        "schemaVersion": "v5.2",
4        "hoverflyVersion": "v1.2.0",
5        "timeExported": "2020-04-25T17:56:32+03:00"
```

See also:

For a hands-on tutorial of creating and editing simulations, see *Creating and exporting a simulation*.

2.2.5 Matching strategies

Hoverfly has two matching strategies. Each has advantages and trade-offs.

Note: In order to fully understand Hoverfly's matching strategies, it is recommended that you read the *Simulations* section first.

Strongest Match

This is the default matching strategy for Hoverfly. If Hoverfly finds multiple Request Response Pairs that match an incoming request, it will return the Response from the pair which has the highest **matching score**.

To set "strongest" as the matching strategy, simply run:

```
hoverctl mode simulate
```

Or to be explicit run:

```
hoverctl mode simulate --matching-strategy=strongest
```

Matching scores

This example shows how matching scores are calculated.

Let's assume Hoverfly is running in simulate mode, and the simulation data contains four *Request Responses Pairs*. Each Request Response Pair contains one or more *Request Matchers*.

Hoverfly then receives a `GET` request to the destination `www.destination.com`. The incoming request contains the following fields.

Request

Field	Value
method	GET
destination	www.destination.com

Request Response Pair 1

Field	Matcher Type	Value	Score	Total Score	Matched?
method	exact	DELETE	+0	1	false
destination	exact	www.destination.com	+1		

This pair contains two Request Matchers. The **method** value in the incoming request (`GET`) does not match the value for the **method** matcher (`DELETE`). However the **destination** value does match.

This gives the Request Response Pair a total score of 1, but since one match failed, it is treated as unmatched (**Matched?** = `false`).

Request Response Pair 2

Field	Matcher Type	Value	Score	Total Score	Matched?
method	exact	GET	+1	1	true

This pair contains one Request Matcher. The **method** value in the incoming request (`GET`) matches the value for the **method** matcher. This gives the pair a total score of 1, and since no matches failed, it is treated as matched.

Request Response Pair 3

Field	Matcher Type	Value	Score	Total Score	Matched?
method	exact	GET	+1	2	true
destination	exact	www.destination.com	+1		

In this pair, the **method** and **destination** values in the incoming request both match the corresponding Request Matcher values. This gives the pair a total score of 2, and it treated as matched.

Request Response Pair 4

Field	Matcher Type	Value	Score	Total Score	Matched?
method	exact	GET	+1	1	false
destination	exact	www.miss.com	+0		

This pair is treated as unmatched because the **destination** matcher failed.

Request Response Pair 3 has the highest score, and is therefore the **strongest match**.

This means that Hoverfly will return the Response contained within Request Response Pair 3.

Note: When there are multiple matches all with the same score, Hoverfly will pick the last one in the simulation.

The strongest match strategy makes it much easier to identify why Hoverfly has not returned a Response to an incoming Request. If Hoverfly is not able to match an incoming Request to a Request Response Pair, it will return the closest match. For more information see *Troubleshooting*.

However, the additional logic required to calculate matching scores does affect Hoverfly's performance.

First Match

First match is the alternative (legacy) mechanism of matching. There is no scoring, and Hoverfly simply returns the first match it finds in the simulation data.

To set first match as the matching strategy, run:

```
hoverctl mode simulate --matching-strategy=first
```

The main advantage of this strategy is performance - although it makes debugging matching errors harder.

2.2.6 Caching

In *Simulate mode*, Hoverfly uses caching in order to retain strong performance characteristics, even in the event of complex matching. The cache is a key-value store of request hashes (hash of all the request excluding headers) to responses.

Caching matches

When Hoverfly receives a request in simulate mode, it will first hash it and then look for it in the cache. If a cache entry is found, it will send the cached response back to Hoverfly. If it is not found, it will look for a match in the list of matchers. Whenever a new match is found, it will be added to the cache.

Caches misses

Hoverfly also caches misses. This means that repeating a request which was not matched will return a cached miss, avoiding the need to perform matching. The closest miss is also cached, so Hoverfly will not lose any useful information about which matchers came closest.

Header caching

Currently, headers are not included in the hash for a request. This is because headers tend to change across time and clients, impacting ordinary and eager caching respectively.

Eager caching

The cache is automatically pre-populated whenever switching to simulate mode. This only works on certain matchers (such as matchers where every field is an "exactMatch"), but it means the initial cache population does not happen **during** simulate mode.

Cache invalidation

Cache invalidation is a straightforward process in Hoverfly. It only occurs when a simulation is modified.

2.2.7 Templating

Hoverfly can build responses dynamically through templating. This is particularly useful when combined with loose matching, as it allows a single matcher to represent an unlimited combination of responses.

Enabling Templating

By default templating is disabled. In order to enable it, set the `templated` field to true in the response of a simulation.

Getting data from the request

Currently, you can get the following data from request to the response via templating:

Field	Example	Request	Result
Request scheme	`{{ Request.Scheme }}`	http://www.foo.com	http
Query parameter value	`{{ Request.QueryParam.myParam }}`	http://www.foo.com?myParam=bar	bar
Query parameter value (list)	`{{ Request.QueryParam. NameOfParameter.[1] }}`	http://www.foo.com?myParam= bar1&myParam=bar2	bar2
Path parameter value	`{{ Request.Path.[1] }}`	http://www.foo.com/zero/one/two	one
Method	`{{ Request.Method }}`	http://www.foo.com/zero/one/two	GET
Host	`{{ Request.Host }}`	http://www.foo.com/zero/one/two	www.foo.com
jsonpath on body	`{{ Request.Body 'jsonpath' '$.id' }}`	{ "id": 123, "username": "hoverfly" }	123
xpath on body	`{{ Request.Body 'xpath' '/ root/id' }}`	<root><id>123</id></root>	123
From data	`{{ Request.FormData.email }}`	email=foo@bar.com	foo@bar.com
Header value	`{{ Request.Header.X-Header-Id }}`	{ "X-Header-Id": ["bar"] }	bar
Header value (list)	`{{ Request.Header.X-Header-Id. [1] }}`	{ "X-Header-Id": ["bar1", "bar2"] }	bar2
State	`{{ State.basket }}`	State Store = {"basket":"eggs"}	eggs

Helper Methods

Additional data can come from helper methods. These are the ones Hoverfly currently support:

Description	Example	Result
The current date time with offset, in the given format. For example: • The current date time plus 1 day in unix timestamp • The current date time in ISO 8601 format • The current date time minus 1 day in custom format	• `{{ now '1d' 'unix' }}` • `{{ now '' '' }}` • `{{ now '-1d' '2006-Jan-02' }}`	• 1136300645 • 2006-01-02T15:04:05Z • 2006-Jan-01
A random string	`{{ randomString }}`	hGfclKjnmwcCds
A random string with a specified length	`{{ randomStringLength 2 }}`	KC
A random boolean	`{{ randomBoolean }}`	true
A random integer	`{{ randomInteger }}`	42
A random integer within a range	`{{ randomIntegerRange 1 10 }}`	7
A random float	`{{ randomFloat }}`	42
A random float within a range	`{{ randomFloatRange 1.0 10.0 }}`	7.4563213423
A random email address	`{{ randomEmail }}`	LoriStewart@Photolist.com
A random IPv4 address	`{{ randomIPv4 }}`	224.36.27.8
A random IPv6 address	`{{ randomIPv6 }}`	41d7:daa0:6e97:6fce:411e:681:f86f:e557
A random UUID	`{{ randomUuid }}`	7b791f3d-d7f4-4635-8ea1-99568d821562
Generate random data using go-fakeit	`{{ faker 'Name' }}`	John Smith
Query CSV data source where ID = 3 and return its name	`{{csv 'test-csv' 'id' '3' 'name'}}`	John Smith
Query Journal index where index value = 1 and return Name from associated Response body in journal entry.	`{{journal "Request.QueryParam.id" "1" "response" "jsonpath" "$.name"}}`	John Smith

Time offset

When using template helper method `now`, time offset must be formatted using the following syntax.

Shorthand	Type
ns	Nanosecond
us/μs	Microsecond
ms	Millisecond
s	Second
m	Minute
h	Hour
d	Day
y	Year

Prefix an offset with – to subtract the duration from the current date time.

Example time offset

5m	5 minutes
1h30m	1 hour 5 minutes
1y10d	1 year 10 days

Date time formats

When using template helper method `now`, date time formats must follow the Golang syntax. More can be found out here https://golang.org/pkg/time/#Parse

Example date time formats

2006-01-02T15:04:05Z07:00
Mon, 02 Jan 2006 15:04:05 MST
Jan _2 15:04:05

Note: If you leave the format string empty, the default format to be used is ISO 8601 (2006-01-02T15:04:05Z07:00).

You can also get an UNIX timestamp by setting the format to:

- `unix`: UNIX timestamp in seconds
- `epoch`: UNIX timestamp in milliseconds

Faker

Support for [go-fakeit](#) was added in order to extend the templating capabilities of Hoverfly. Faker covers many different test data requirements and it can be used within Hoverfly templated responses by using the `faker` helper followed by the faker type (e.g. `Name`, `Email`) For example, you can generate a random name using the following expression:

```
{
    "body": "{\"name\": \"{{faker 'Name'}}\"}"
}
```

Fakers that require arguments are currently not supported.

CSV Data Source

You can query data from a CSV data source.

```
{
    "body": "{\"name\": \"{{csv '(data-source-name)' '(column-name)' '(query-value)'
→'(selected-column)' }}\"}"
}
```

Note: The data source name is case sensitive whereas other parameters in this function are case insensitive. You can use hoverctl or call the Admin API to upload CSV data source to a running Hoverfly instance.

Example: Start Hoverfly with a CSV data source (student-marks.csv) provided below.

```
hoverfly -templating-data-source "student-marks <path to below CSV file>"
```

ID	Name | Marks
1	Test1 | 55
2	Test2 | 65
3	Test3 | 98
4	Test4 | 23
5	Test5 | 15
•	NA | 0

Description	Example	Result
Search where ID = 3 and return name	csv 'student-marks' 'Id' '3' 'Name'	Test3
Search where ID = 4 and return its marks	csv 'student-marks' 'Id' '4' 'Marks'	Test23
Search where Name = Test1 and return marks	csv 'student-marks' 'Name' 'Test1' 'Marks'	55
Search where Id is not match and return marks (in this scenario, it matches wildcard * and returns)	csv 'student-marks' 'Id' 'Test100' 'Marks'	0
Search where Id = first path param and return marks URL looks like - http://test.com/students/5/marks	csv 'student-marks' 'Id' 'Request.Path.[0]' 'Marks'	15

Journal Entry Data

Journal Entry can be queried using its index and its extracted value.

Syntax

```
journal "index name" "extracted value" "request/response" "xpath/jsonpath" "lookup query"
```

`index name` should be the same key expression you have specified when you enable the journal index. `extracted value` is for doing a key lookup for the journal entry from that index. `request/response` specifies if you want to get data from the request or response. `xpath/jsonpath` specifies whether you want to extract it using xpath or json path expression. `lookup query` is either jsonpath or xpath expressions to parse the request/response data.

Example:

```
{
    "body": "{\"name\": \"{{ journal 'Request.QueryParam.id' '1' 'response' 'jsonpath' '
↪$.name' }}\"}"
}
```

In the above example, we are querying the name from JSON response in the journal entry where index `Request.QueryParam.id` has a key value of 1.

Key Value Data Store

Sometimes you may need to store a temporary variable and retrieve it later in other part of the templated response. In this case, you can use the internal key value data store. The following helper methods are available:

Description	Example	Result
Put an entry	`{{ putValue 'id' 123 true }}`	123
Get an entry	`{{ getValue 'id' }}`	123
Add a value to an arra	`{{ addToArray 'names' 'John' true }}`	John
Get an array	`{{ getArray 'names' }}`	[]string{"John"

`addToArray` will create a new array if one doesn't exist. The boolean argument in `putValue` and `addToArray` is used to control whether the set value is returned.

Note: Each templating session has its own key value store, which means all the data you set will be cleared after the current response is rendered.

Maths Operation

The basic maths operations are currently supported: add, subtract, multiply and divide. These functions take three parameters: two values it operates on and the precision. The precision is given in a string format such as `'0.00'`. For example `{{ add 3 2.5 '0.00' }}` should give you `5.50`. If no format is given, the exact value will be printed with up to 6 decimal places.

Description	Example	Result
Add	`{{ add 10 3 '0.00' }}`	13.33
Subtract	`{{ subtract 10 3 '' }}`	7
Multiply	`{{ multiply 10 3 '' }}`	30
Divide	`{{ divide 10 3 '' }}`	3.333333

A math functions for summing an array of numbers is also supported; it's usually used in conjunction with the `#each` block helper. For example:

With the request payload of

```
{
    "lineitems": {
        "lineitem": [
            {
                "upc": "1001",
                "quantity": "1",
                "price": "3.50"
            },
            {
                "upc": "1002",
                "quantity": "2",
                "price": "4.50"
            }
        ]
    }
}
```

We can get the total price of all the line items using this templating function:

```
{{#each (Request.Body 'jsonpath' '$.lineitems.lineitem') }}      {{ addToArray 'subtotal'
(multiply (this.price) (this.quantity) '') false }} {{/each}}     total:  {{ sum (getArray
'subtotal') '0.00' }}
```

String Operation

You can use the following helper methods to join, split or replace string values.

Description	Example	Result
String concatenate	`{{ concat 'bee' 'hive' }}`	beehive
String splitting	`{{ split 'bee,hive' ',' }}`	[]string{"bee", "hive"}
Replace all occurrences of the old value with the new value in the target string	`{{ replace (Request.Body 'jsonpath' '$.text')` `'be' 'mock' }}` (where Request.Body has the value of `{"text":"to be or not to be"}`	to mock or not to mock

Conditional Templating, Looping and More

Hoverfly uses the https://github.com/aymerick/raymond library for templating, which is based on http://handlebarsjs.com/

To learn about more advanced templating functionality, such as looping and conditionals, read the documentation for these projects.

Global Literals and Variables

You can define global literals and variables for templated response. This comes in handy when you have a lot of templated responses that share the same constant values or helper methods.

Literals

Literals are constant values. You can declare literals as follows and then reference it in templated response as `{{ Literals.<literal name> }}`.

```
{
  "data": {
  ...
  "literals": [
       {
           "name":"literal1",
           "value":"value1"
       },
       {
           "name":"literal2",
           "value":["value1", "value2", "value3"]
       },
       {
           "name":"literal3",
           "value": {
               "key": "value"
           }
       }
    ]
}
```

Variables

Variable lets you define a helper method that can be shared among templated responses. You can associate the helper method with a name and then reference it in templated response as `{{ Vars.<variable name> }}`.

```
{
  "data": {
  ...
  "variables": [
       {
           "name":"<variable name>",
           "function":"<helper method name>",
```

```
            "arguments":["arg1", "arg2"]

        }
    ]
}

{
  "data": {
  ...
  "variables": [
        {
            "name":"varOne",
            "function":"faker",
            "arguments":["Name"]

        },
        {

            "name":"idFromJSONRequestBody",
            "function":"requestBody",
            "arguments":["jsonpath", "$.id"]
        },
        {

            "name":"idFromXMLRequestBody",
            "function":"requestBody",
            "arguments":["xpath", "/root/id"]
        }
    ]
}
```

2.2.8 State

Hoverfly contains a map of keys and values which it uses to store it's internal state. Some *Request matchers* can be made to only match when Hoverfly is in a certain state, and other matchers can be set to mutate Hoverfly's state.

Setting State when Performing a Match

A response includes two fields, *transitionsState* and *removesState* which alter Hoverflies internal state during a match:

```
"request": {
    "path": [
        {
            "matcher": "exact",
            "value": "/pay"
        }
    ]
},
"response": {
    "status": 200,
    "body": "eggs and large bacon",
    "transitionsState" : {
```

```
        "payment-flow" : "complete"
    },
    "removesState" : [
        "basket"
    ]
}
```

In the above case, the following changes to Hoverflies internal state would be made on a match:

Current State of Hoverfly	New State of Hoverfly?	reason
payment-flow=pending,basket=full	payment-flow=complete	Payment value transitions, basket deleted by key
basket=full	payment-flow=complete	Payment value created, basket deleted by key
	payment-flow=complete	Payment value created, basket already absent

Requiring State in order to Match

A matcher can include a field *requiresState*, which dictates the state Hoverfly must be in for there to be a match:

```
"request": {
    "path": [
        {
            "matcher": "exact",
            "value": "/basket"
        }
    ]
    "requiresState": {
        "eggs": "present",
        "bacon" : "large"
    }
},
"response": {
    "status": 200,
    "body": "eggs and large bacon"
}
```

In the above case, the following matches results would occur when making a request to *basket*:

Current State of Hoverfly	matches?	reason
eggs=present,bacon=large	true	Required and current state are equal
eggs=present,bacon=large,f=x	true	Additional state 'f=x' is not used by this matcher
eggs=present	false	Bacon is missing
eggs=present,bacon=small	false	Bacon is has the wrong value

Managing state via Hoverctl

It could be tricky to reason about the current state of Hoverfly, or to get Hoverfly in a state that you desire for testing. This is why Hoverctl comes with commands that let you orchestrate it's state. Some useful commands are:

```
$ hoverctl state --help
$ hoverctl state get-all
$ hoverctl state get key
$ hoverctl state set key value
$ hoverctl state delete-all
```

Sequences

Using state, it is possible to recreate a sequence of different responses that may come back given a single request. This can be useful when trying to test stateful endpoints.

When defining state for request response pairs, if you prefix your state key with the string `sequence:`, Hoverfly will acknowledge the pair as being part of a stateful sequence. When simulating this sequence, Hoverfly will keep track of the user's position in the sequence and move them forwards.

Once Hoverfly has reached the end of the sequence, it will continue to return the final response.

```
{
  "data": {
    "pairs": [{
        "request": {
          "requiresState": {
            "sequence:1": "1"
          }
        },
        "response": {
          "status": 200,
          "body": "First response",
          "transitionsState": {
            "sequence:1": "2"
          }
        }
      },
      {
        "request": {
          "requiresState": {
            "sequence:1": "2"
          }
        },
        "response": {
          "status": 200,
          "body": "Second response"
        }
      }
    ]
  },
  "meta": {
    "schemaVersion": "v5.2"
```

```
    }
}
```

2.2.9 Destination filtering

By default, Hoverfly will process every request it receives. However, you may wish to control which URLs Hoverfly processes.

This is done by *filtering* the *destination* URLs using either a string or a regular expression. The *destination* string or regular expression will be compared against the host and the path of a URL.

For example, specifying `hoverfly.io` as the destination value will tell Hoverfly to process only URLs on the `hoverfly.io` host.

```
hoverctl destination "hoverfly.io"
```

Specifying `api` as the *destination* value during *Capture mode* will tell Hoverfly to capture only URLs that contain the string `api`. This would include both `api.hoverfly.io/endpoint` and `hoverfly.io/api/endpoint`.

See also:

This functionality is best understood via a practical example: see *Capturing or simulating specific URLs* in the *Tutorials* section.

Note: The destination setting applies to all Hoverfly modes. If a destination value is set while Hoverfly is running in *Simulate mode*, requests that are excluded by the destination setting will be passed through to the real URLs. This makes it possible to return both real and simulated responses.

2.2.10 Middleware

Middleware intercepts traffic between the client and the API (whether real or simulated), and allowing you to manipulate it.

You can use middleware to manipulate data in simulated responses, or to inject unpredictable performance characteristics into your simulation.

Middleware works differently depending on the Hoverfly mode.

- Capture mode: middleware affects only **outgoing requests**

- Simulate/Spy mode: middleware affects only **responses** (cache contents remain untouched)

- Synthesize mode: middleware **creates responses**

- Modify mode: middleware affects **requests and responses**

Note: Middleware is applied after rendering the templating functions (see *Templating*) in the response body.

You can write middleware in any language. There are two different types of middleware.

Local Middleware

Hoverfly has the ability to invoke middleware by executing a script or binary file on a host operating system. The only requires are that the provided middleware can be executed and sends the Middleware JSON schema to stdout when the Middleware JSON schema is received on stdin.

HTTP Middleware

Hoverfly can also send middleware requests to a HTTP server instead of running a process locally. The benefits of this are that Hoverfly does not initiate the process, giving more control to the user. The only requirements are that Hoverfly can POST the Middleware JSON schema to middleware URL provided and the middleware HTTP server responses with a 200 and the Middleware JSON schema is in the response.

Middleware Interface

When middleware is called by Hoverfly, it expects to receive and return JSON (see *Simulation schema*). Middleware can be used to modify the values in the JSON but **must not** modify the schema itself.
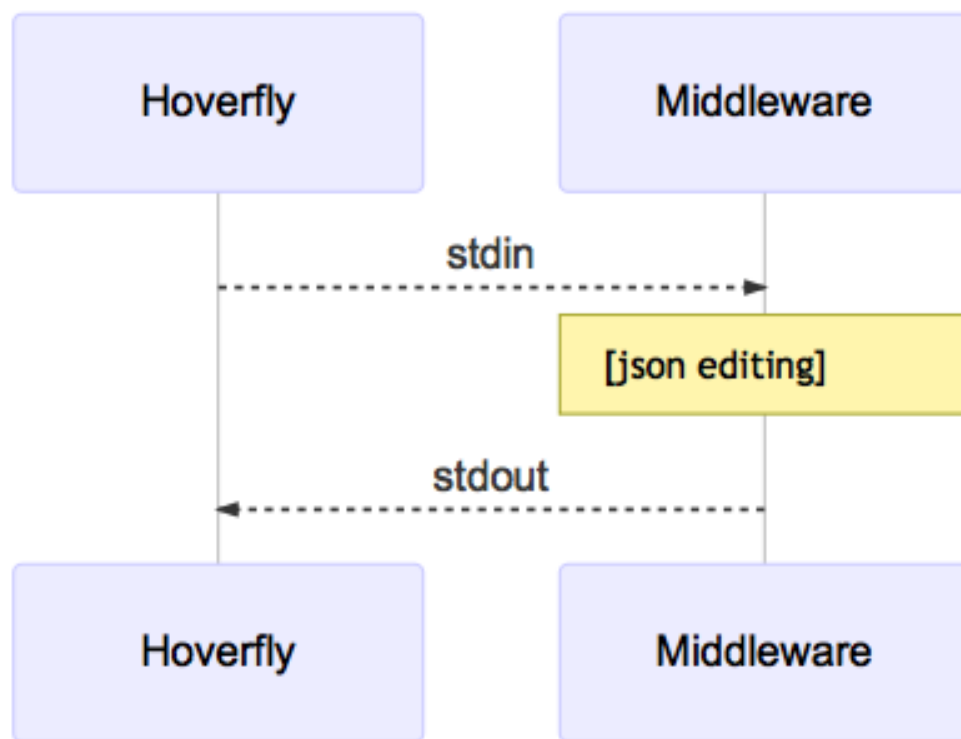
Hoverfly will send the JSON object to middleware via the standard input stream. Hoverfly will then listen to the standard output stream and wait for the JSON object to be returned.

See also:

Middleware examples are covered in the tutorials section. See *Using middleware to simulate network latency* and *Using middleware to modify response payload and status code*.

2.2.11 Post Serve Action

Overview

- PostServeAction allows you to execute custom code or invoke endpoint with request-response pair after a response has been served in simulate or spy mode.

- It is custom script that can be written in any language. Hoverfly has the ability to invoke a script or binary file on a host operating system or on the remote host. Custom code is executed/remote host is invoked after a provided delay(in ms) once simulated response is served.

- We can register multiple post serve actions.

- In order to register local post serve action, it takes mainly four parameters - binary to invoke script, script content/location, delay(in ms) post which it will be executed and name of that action.

- In order to register remote post serve action, it takes mainly three parameters - remote host to be invoked with request-response pair, delay(in ms) post which it will be executed and name of that action.

Ways to register a Post Serve Action

- At time of startup by passing single/multiple -post-serve-action flag(s) as mentioned in the hoverfly command page.

- Via PUT API to register new post serve action as mentioned in the API page.

- Using command line hoverctl post-serve-action set command as mentioned in the hoverctl command page.

- Once post serve action is registered, we can trigger particular post serve action by putting it in response part of request-response pair in simulation JSON.

Example Simulation JSON

```
{
    "response": {
        "postServeAction": "<name of post serve action we want to invoke>"
    }
}
```

2.2.12 Hoverctl

Hoverctl is a command line utility that is shipped with Hoverfly. Its purpose is to make it easier to interact with Hoverfly APIs and your local filesystem, while also providing a way to start and stop instances of Hoverfly.

Hoverctl also has the ability to work with multiple instances of Hoverfly, through the use of the target option. Configuration is stored against each target meaning it is possible to start an instance of Hoverfly locally and remotely and still be able to interact with each separately.

See also:

Please refer to *hoverctl commands* for more information about hoverctl.

See also:

Please refer to the *Controlling a remote Hoverfly instance with hoverctl* tutorial for a step by step example of using targets.

2.3 Native language bindings

Native language bindings are available for Hoverfly to make it easy to integrate into different environments.

2.3.1 HoverPy

To get started:

```
sudo pip install hoverpy
python
```

And in Python you can simply get started with:

```python
import hoverpy
import requests

# capture mode
with hoverpy.HoverPy(capture=True) as hp:
    data = requests.get("http://time.jsontest.com/").json()

# simulation mode
with hoverpy.HoverPy() as hp:
    simData = requests.get("http://time.jsontest.com/").json()
    print(simData)

assert(data["milliseconds_since_epoch"] == simData["milliseconds_since_epoch"])
```

For more information, read the HoverPy documentation.

2.3.2 Hoverfly Java

- Strict or loose HTTP request matching based on URL, method, body and header combinations
- Fluent and expressive DSL for easy generation of simulated APIs
- Automatic marshalling of objects into JSON during request/response body generation
- HTTPS automatically supported, no extra configuration required
- Download via Maven or Gradle

To get started, read the Hoverfly Java documentation.

2.4 Tutorials

In these examples, we will use the hoverctl (CLI tool for Hoverfly) to interact with Hoverfly.

Hoverfly can also be controlled via its *REST API*, or via *Native language bindings*.

2.4.1 Basic tutorials

Creating and exporting a simulation

Note: If you are running Hoverfly on a machine that accesses the internet via a proxy (for example if you are on a corporate network), please follow the *Using Hoverfly behind a proxy* tutorial before proceeding.

Start Hoverfly and set it to Capture mode

```
hoverctl start
hoverctl mode capture
```

Make a request with cURL, using Hoverfly as a proxy server:

```
curl --proxy http://localhost:8500 http://time.jsontest.com
```

View the Hoverfly logs

```
hoverctl logs
```

Export the simulation to a JSON file

```
hoverctl export simulation.json
```

Stop hoverfly

```
hoverctl stop
```

You'll now see a `simulation.json` file in your current working directory, which contains all your simulation data.

In case you are curious, the sequence diagram for this process looks like this:

Note: By default, request headers are not captured. If you want to capture headers, you will need to specify them when setting capture mode.

```
hoverctl mode capture --headers "User-Agent,Content-Type,Authorization"
hoverctl mode capture --all-headers
```

Note: It is possible to filter and export simulation to separate JSON files by providing a plain text or regex string to the `--url-pattern` flag:

```
hoverctl export echo.json --url-pattern "echo.jsontest.com"     // export simulations␣
→for echo.jsontest.com only
hoverctl export api.json --url-pattern "(.+).jsontest.com"      // export simulations␣
→for all jsontest.com subdomains
```
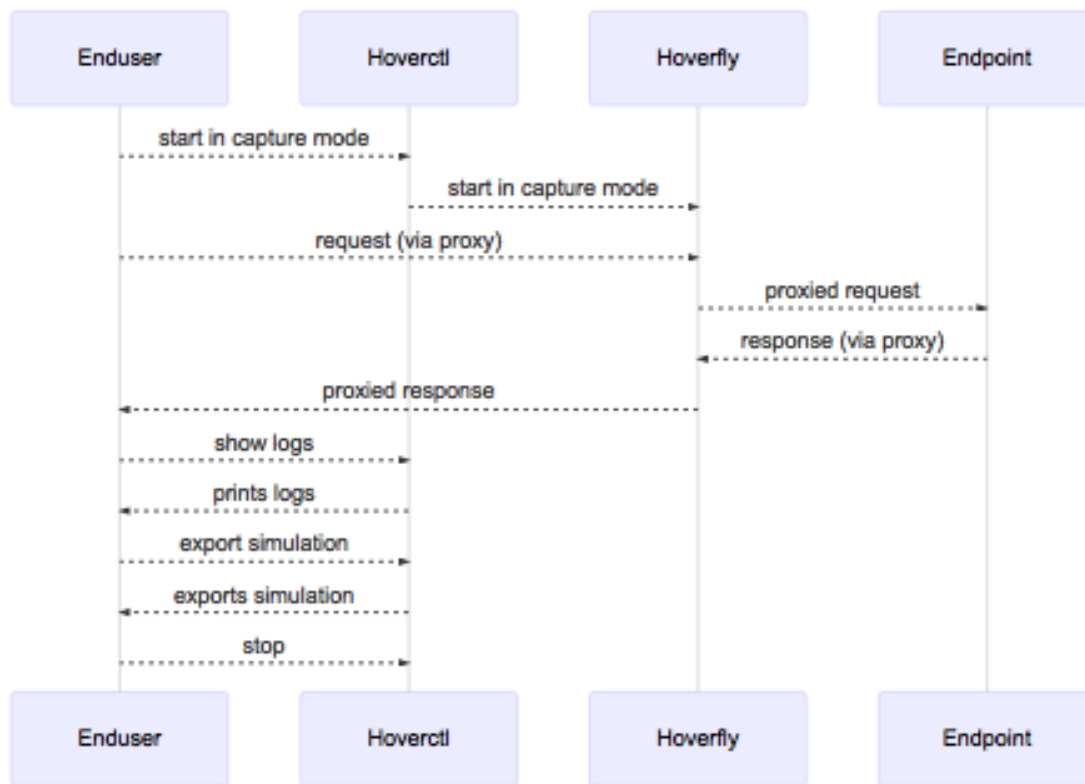
Importing and using a simulation

In this tutorial we are going to import the simulation we created in the previous tutorial.

```
hoverctl start
hoverctl import simulation.json
```

Hoverfly can also import simulation data that is stored on a remote host via HTTP:

```
hoverctl import https://example.com/example.json
```

Make a request with cURL, using Hoverfly as a proxy.

```
curl --proxy localhost:8500 http://time.jsontest.com
```

This outputs the time at the time the request was captured.

```
{
    "time": "02:07:28 PM",
    "milliseconds_since_epoch": 1482242848562,
    "date": "12-20-2016"
}
```

Stop Hoverfly:

```
hoverctl stop
```

Note: Importing multiple simulations:

The above example shows importing one simulation into Hoverfly. You can also import multiple simulations:

```
hoverctl simulation add foo.json bar.json
```

You can specify one or more simulations when starting Hoverfly using `hoverfly` or `hoverctl` commands:

```
hoverfly -import foo.json -import bar.json
hoverctl start --import foo.json --import bar.json
```

Hoverfly appends any unique pair to the existing simulation by comparing the equality of the request JSON objects. If a conflict occurs, the pair is not added.

Adding delays to a simulation

Simulating API latency during development allows you to write code that will deal with it gracefully.

In Hoverfly, this is done by applying "delays" or "delaysLogNormal" to responses in a simulation.

Delays are applied by editing the Hoverfly simulation JSON file. Delays can be applied selectively according to request URL pattern and/or HTTP method.

Applying a delay to all responses

Let's apply a 2 second delay to all responses. First, we need to create and export a simulation.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 http://time.jsontest.com
curl --proxy localhost:8500 http://date.jsontest.com
hoverctl export simulation.json
hoverctl stop
```

Take a look at the `"globalActions"` property within the `simulation.json` file you exported. It should look like this:

```
1      "globalActions": {
2          "delays": [],
3          "delaysLogNormal": []
```

Edit the file so the `"globalActions"` property looks like this:

```
1      "globalActions": {
2          "delays": [
3              {
4                  "urlPattern": ".",
5                  "httpMethod": "",
6                  "delay": 2000
7              }
8          ],
9          "delaysLogNormal": []
```

Hoverfly will apply a delay of 2000ms to all URLs that match the `"urlPattern"` value. We want the delay to be applied to **all URLs**, so we set the `"urlPattern"` value to the regular expression `"."`.

Now import the edited `simulation.json` file, switch Hoverfly to Simulate mode and make the requests again.

```
hoverctl start
hoverctl import simulation.json
curl --proxy localhost:8500 http://time.jsontest.com
curl --proxy localhost:8500 http://date.jsontest.com
hoverctl stop
```

The responses to both requests are delayed by 2 seconds.

Applying a delay to specific responses

When you want to add a delay to specific responses and global host / uri / http method matching is not the case, you can specify **fixedDelay** in response. Here we apply a delay of *3000ms* only to request **/api/profile** with **X-API-Version: v1** header:

```
{
    "request": {
        "path": [
            {"matcher": "exact", "value": "/api/profile"}
        ],
        "headers": {
            "X-API-Version": [
                {"matcher": "exact", "value": "v1"}
            ]
        }
    },
    "response": {
        "status": 404,
        "body": "Page not found",
        "fixedDelay": 3000
    }
}
```

It's also possible to apply a logNormal delay:

```
{
    "request": {
        "path": [
            {"matcher": "exact", "value": "/api/profile"}
        ],
        "headers": {
            "X-API-Version": [
                {"matcher": "exact", "value": "v1"}
            ]
        }
    },
    "response": {
        "status": 404,
        "body": "Page not found",
```

```
        "logNormalDelay": {
            "min": 100,
            "max": 10000,
            "mean": 5000,
            "median": 500
        }
    }
}
```

Like global delays, when both *fixedDelay* and *logNormalDelay* are provided they are applied one after another.

Applying different delays based on host

Now let's apply a delay of 1 second on responses from `time.jsontest.com` and a delay of 2 seconds on responses from `date.jsontest.com`.

Run the following to create and export a simulation.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 http://time.jsontest.com
curl --proxy localhost:8500 http://date.jsontest.com
hoverctl export simulation.json
hoverctl stop
```

Edit the `simulation.json` file so that the `"globalActions"` property looks like this:

```
1      "globalActions": {
2          "delays": [
3              {
4                  "urlPattern": "time\\.jsontest\\.com",
5                  "httpMethod": "",
6                  "delay": 1000
7              },
8              {
9                  "urlPattern": "date\\.jsontest\\.com",
10                 "httpMethod": "",
11                 "delay": 2000
12             }
13         ],
14         "delaysLogNormal": []
```

Now run the following to import the edited `simulation.json` file and run the simulation:

```
hoverctl start
hoverctl import simulation.json
curl --proxy localhost:8500 http://time.jsontest.com
curl --proxy localhost:8500 http://date.jsontest.com
hoverctl stop
```

You should notice a 1 second delay on responses from `time.jsontest.com`, and a 2 second delay on responses from `date.jsontest.com`.

Note: You can easily get into a situation where your request URL has multiple matches. In this case, the first successful match wins.

Applying different delays based on URI

Now let's apply different delays based on location. Run the following to create and export a simulation.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 http://echo.jsontest.com/a/b
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl --proxy localhost:8500 http://echo.jsontest.com/c/d
hoverctl export simulation.json
hoverctl stop
```

Edit the `simulation.json` file so that the `"globalActions"` property looks like this:

```
1      "globalActions": {
2          "delays": [
3              {
4                  "urlPattern": "echo\\.jsontest\\.com\\/a\\/b",
5                  "httpMethod": "",
6                  "delay": 2000
7              },
8              {
9                  "urlPattern": "echo\\.jsontest\\.com\\/b\\/c",
10                 "httpMethod": "",
11                 "delay": 2000
12             },
13             {
14                 "urlPattern": "echo\\.jsontest\\.com\\/c\\/d",
15                 "httpMethod": "",
16                 "delay": 3000
17             }
18         ],
19         "delaysLogNormal": []
```

Now run the following to import the edited `simulation.json` file and run the simulation:

```
hoverctl start
hoverctl import simulation.json
hoverctl mode simulate
curl --proxy localhost:8500 http://echo.jsontest.com/a/b
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl --proxy localhost:8500 http://echo.jsontest.com/c/d
hoverctl stop
```

You should notice a 2 second delay on responses from `echo.jsontest.com/a/b` and `echo.jsontest.com/b/c`, and a 3 second delay on the response from `echo.jsontest.com/c/d`.

Note: When you use *Hoverfly as a webserver*, you should replace or remove any original host pattern

in the `urlPattern` filter, because the endpoint URL is based on Hoverfly webserver address (eg. `http://localhost:8888`).

Applying different delays based on HTTP method

Let's apply a delay of 2 seconds on responses to **GET requests only** made to `echo.jsontest.com/b/c`.

Run the following to create and export a simulation.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl --proxy localhost:8500 -X POST http://echo.jsontest.com/b/c
hoverctl export simulation.json
hoverctl stop
```

Edit the `simulation.json` file so that the `"globalActions"` property looks like this:

```
1       "globalActions": {
2           "delays": [
3               {
4                   "urlPattern": "echo\\.jsontest\\.com\\/b\\/c",
5                   "httpMethod": "GET",
6                   "delay": 2000
7               }
8           ],
9           "delaysLogNormal": []
```

Now run the following to import the edited `simulation.json` file and run the simulation:

```
hoverctl start
hoverctl import simulation.json
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl -X POST --proxy localhost:8500 http://echo.jsontest.com/b/c
hoverctl stop
```

You should notice a 2 second delay on the response to the GET request and no delay on the response to the POST request.

Log-normal distributions of delay

The Log-normal distribution is a pretty accurate description of a server latency. The Log-normal distribution defines by 2 parameters and . We will compute these parameters from mean and median of a server response time. These values you can see in your monitoring of the production server. If need you can adjust response time by min and max parameters.

Let's apply a random log-normal distributed delay to all responses. First, we need to create and export a simulation.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl --proxy localhost:8500 -X POST http://echo.jsontest.com/b/c
```

```
hoverctl export simulation.json
hoverctl stop
```

Take a look at the `"globalActions"` property within the `simulation.json` file you exported. It should look like this:

```
1       "globalActions": {
2           "delays": [],
3           "delaysLogNormal": []
```

Edit the file so the `"globalActions"` property looks like this:

```
1        "globalActions": {
2            "delaysLogNormal": [
3                {
4                    "urlPattern": ".",
5                    "httpMethod": "",
6                    "min": 100,
7                    "max": 10000,
8                    "mean": 5000,
9                    "median": 500
10               }
11           ]
12       }
```

Hoverfly will apply a delay to all URLs that match the `"urlPattern"` value. We want the delay to be applied to **all URLs**, so we set the `"urlPattern"` value to the regular expression `"."`.

Now import the edited `simulation.json` file, switch Hoverfly to Simulate mode and make the requests again.

```
hoverctl start
hoverctl import simulation.json
curl --proxy localhost:8500 http://echo.jsontest.com/b/c
curl -X POST --proxy localhost:8500 http://echo.jsontest.com/b/c
hoverctl stop
```

Loose request matching using a Request Matcher

See also:

Please carefully read through *Request Responses Pairs* alongside this tutorial to gain a high-level understanding of what we are about to cover.

In some cases you may want Hoverfly to return the same stored response for more than one incoming request. This can be done using Request Matchers.

Let's begin by capturing some traffic and exporting a simulation. This step saves us having to manually create a simulation ourselves and gives us a request to work with.

```
hoverctl start
hoverctl mode capture
curl --proxy http://localhost:8500 http://echo.jsontest.com/foo/baz/bar/spam
hoverctl export simulation.json
hoverctl stop
```

If you take a look at your `simulation.json` you should notice these lines in your request.

```
"path": [
        {
            "matcher": "exact",
            "value": "/foo/baz/bar/spam"
        }
 ]
```

Modify them to:

```
"path": [
        {
            "matcher": "glob",
            "value": "/foo/*/bar/spam"
        }
 ]
```

Save the file as `simulationimport.json` and run the following command to import it and cURL the simulated endpoint:

```
hoverctl start
hoverctl mode simulate
hoverctl import simulationimport.json
curl --proxy http://localhost:8500 http://echo.jsontest.com/foo/QUX/bar/spam
hoverctl stop
```

The same response is returned, even though we created our simulation with a request to `http://echo.jsontest.com/foo/baz/bar/spam` in Capture mode and then sent a request to `http://echo.jsontest.com/foo/QUX/bar/spam` in Simulate mode.

See also:

In this example we used the `globMatch` Request Matcher type. For a list of other Request Matcher types and examples of how to use them, please see the *Request matchers* section.

Note: Key points:

- To change how incoming requests are matched to stored responses, capture a simulation, export it, edit it

- While editing, choose a request field to match on, select a Request Matcher type and a matcher value

- Re-import the simulation

- Requests can be manually added without capturing the request

---

Using middleware to simulate network latency

See also:

Please carefully read through *Middleware* alongside these tutorials to gain a high-level understanding of what we are about to cover.

We will use a Python script to apply a random delay of less than one second to every response in a simulation.

Before you proceed, please ensure that you have Python installed.

Let's begin by writing our middleware. Save the following as `middleware.py`:

```python
#!/usr/bin/env python
import sys
import logging
import random
from time import import sleep

logging.basicConfig(filename='random_delay_middleware.log', level=logging.DEBUG)
logging.debug('Random delay middleware is called')

# set delay to random value less than one second

SLEEP_SECS = random.random()

def main():

    data = sys.stdin.readlines()
    # this is a json string in one line so we are interested in that one line
    payload = data[0]
    logging.debug("sleeping for %s seconds" % SLEEP_SECS)
    sleep(SLEEP_SECS)


    # do not modifying payload, returning same one
    print(payload)

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

The middleware script delays each response by a random value of less than one second.

```
hoverctl start
hoverctl mode capture
curl --proxy http://localhost:8500 http://time.jsontest.com
hoverctl mode simulate
hoverctl middleware --binary python --script middleware.py
curl --proxy http://localhost:8500 http://time.jsontest.com
hoverctl stop
```

Middleware gives you control over the behaviour of a simulation, as well as the data.

Note: Middleware gives you flexibility when simulating network latency - allowing you to randomize the delay value for example - but a new process is spawned every time the middleware script is executed. This can impact Hoverfly's performance under load.

If you need to simulate latency during a load test, it is recommended that you use Hoverfly's native *Delays* functionality to simulate network latency (see *Adding delays to a simulation*) instead of writing middleware. The delays functionality sacrifices flexibility for performance.

Using middleware to modify response payload and status code

See also:

Please carefully read through *Middleware* alongside these tutorials to gain a high-level understanding of what we are about to cover.

We will use a python script to modify the body of a response and randomly change the status code.

Let's begin by writing our middleware. Save the following as `middleware.py`:

```python
#!/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'] = random.choice([200, 201])

    if "response" in payload_dict and "body" in payload_dict["response"]:
        payload_dict["response"]["body"] = "{'foo': 'baz'}\n"

    print(json.dumps(payload_dict))

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

The middleware script randomly toggles the status code between 200 and 201, and changes the response body to a dictionary containing `{'foo':'baz'}`.

```
hoverctl start
hoverctl mode capture
curl --proxy http://localhost:8500 http://time.jsontest.com
hoverctl mode simulate
hoverctl middleware --binary python --script middleware.py
curl --proxy http://localhost:8500 http://time.jsontest.com
hoverctl stop
```

As you can see, middleware allows you to completely modify the content of a simulated HTTP response.

Simulating HTTPS APIs

To capture HTTPS traffic, you need to use Hoverfly's SSL certificate.

First, download the certificate:

```
wget https://raw.githubusercontent.com/SpectoLabs/hoverfly/master/core/cert.pem
```

We can now run Hoverfly with the standard `capture` then `simulate` workflow.

```
hoverctl start
hoverctl mode capture
curl --proxy localhost:8500 https://example.com --cacert cert.pem
hoverctl mode simulate
curl --proxy localhost:8500 https://example.com --cacert cert.pem
hoverctl stop
```

Curl makes the HTTPS request by first establishing a TLS tunnel to the destination with Hoverfly using the HTTP CONNECT method. As curl has supplied Hoverfly's SSL certificate, Hoverfly is then able to intercept and capture the traffic. Effectively SSL-encrypted communication (HTTPS) is established through an unencrypted HTTP proxy.

Note: This example uses cURL. If you are using Hoverfly in another environment, you will need to add the certificate to your trust store. This is done automatically by the Hoverfly Java library (see *Hoverfly Java*).

See also:

This example uses Hoverfly's default SSL certificate. Alternatively, you can use Hoverfly to generate a new certificate. For more information, see *Configuring SSL in Hoverfly*.

Running Hoverfly as a webserver

See also:

Please carefully read through *Hoverfly as a webserver* alongside this tutorial to gain a high-level understanding of what we are about to cover.

Below is a complete example how to capture data with Hoverfly running as a proxy, and how to save it in a simulation file.

```
hoverctl start
hoverctl mode capture
curl --proxy http://localhost:8500 http://echo.jsontest.com/a/b
hoverctl export simulation.json
hoverctl stop
```

Now we can use Hoverfly as a **webserver** in Simulate mode.

```
hoverctl start webserver
hoverctl import simulation.json
curl http://localhost:8500/a/b
hoverctl stop
```

Hoverfly returned a response to our request while running as a webserver, not as a proxy.

Notice that we issued a cURL command to `http://localhost:8500/a/b` instead of `http://echo.jsontest.com/a/b`. This is because when running as a webserver, Hoverfly strips the domain from the endpoint URL in the simulation.

This is explained in more detail in the *Hoverfly as a webserver* section.

Note: Hoverfly starts in Simulate mode by default.

Enable CORS support on Hoverfly

By enabling CORS (Cross-Origin Resource Sharing) support, your web application running on the browser can make requests to Hoverfly even it's not on the same domain.

Starting Hoverfly with CORS enabled is simple:

```
hoverfly -cors
```

Or using *hoverctl*:

```
hoverctl start --cors
```

You can check if CORS is enabled on the Hoverfly by querying the status:

```
hoverctl status
```

When CORS is enabled, Hoverfly intercepts any pre-flight request, and returns an empty 200 response with the following default CORS headers:

- Access-Control-Allow-Origin: (same value as the `Origin` header from the request)
- Access-Control-Allow-Methods: GET,POST,PUT,PATCH,DELETE,HEAD,OPTIONS
- Access-Control-Allow-Headers: (same value as the `Access-Control-Request-Headers` header from the request)
- Access-Control-Max-Age: 1800
- Access-Control-Allow-Credentials: true

Hoverfly also intercepts the actual CORS requests, and add the following default CORS headers to the response:

- Access-Control-Allow-Origin: (same value as the `Origin` header from the request)
- Access-Control-Allow-Credentials: true

Support for customizing the CORS headers will be added in the future release.

Note: Two points to notice when Hoverfly is in capture mode and CORS is enabled: 1. Pre-flight requests handling and CORS headers provided by Hoverfly are not recorded in the simulation. 2. Hoverfly preserves the CORS headers from the remote server if they are present.

Capturing or simulating specific URLs

We can use the `hoverctl destination` command to specify which URLs to capture or simulate. The destination setting can be tested using the `--dry-run` flag. This makes it easy to check whether the destination setting will filter out URLs as required.

```
hoverctl start
hoverctl destination "ip" --dry-run http://ip.jsontest.com
hoverctl destination "ip" --dry-run http://time.jsontest.com
hoverctl stop
```

This tells us that setting the destination to `ip` will allow the URL `http://ip.jsontest.com` to be captured or simulated, while the URL `http://time.jsontest.com` will be ignored.

Now we have checked the destination setting, we can apply it to filter out the URLs we don't want to capture.

```
hoverctl start
hoverctl destination "ip"
hoverctl mode capture
curl --proxy http://localhost:8500 http://ip.jsontest.com
curl --proxy http://localhost:8500 http://time.jsontest.com
hoverctl logs
hoverctl export simulation.json
hoverctl stop
```

If we examine the logs and the `simulation.json` file, we can see that *only* a request response pair to the `http://ip.jsontest.com` URL has been captured.

The destination setting can be either a string or a regular expression.

```
hoverctl start
hoverctl destination "^.*api.*com" --dry-run https://api.github.com
hoverctl destination "^.*api.*com" --dry-run https://api.slack.com
hoverctl destination "^.*api.*com" --dry-run https://github.com
hoverctl stop
```

Here, we can see that setting the destination to `^.*api.*com` will allow the `https://api.github.com` and `https://api.slack.com` URLs to be captured or simulated, while the `https://github.com` URL will be ignored.

Capturing a stateful sequence of responses

By default Hoverfly will store a given request/response pair once only. If the same request returns different responses you may want capture the sequence of changing request/response pairs. You may want to do this because the API is stateful rather than stateless.

A simple example of this is an API that returns the time.

To record a sequence of request/responses where the request is the same but the response is different, we need to enable stateful recording in capture mode.

```
hoverctl start
hoverctl mode capture --stateful
```

Now that we have enabled stateful recording, we can capture several request/response pairs.

```
curl --proxy http://localhost:8500 http://time.jsontest.com

curl --proxy http://localhost:8500 http://time.jsontest.com
```

Once we have finished capturing requests, we can switch Hoverfly back to simulate mode.

```
hoverctl mode simulate
```

Now we are in simulate, we can make the same requests again, and we will see the time update each request we make until we reach the end of our recorded sequence.

```
{
    "time": "01:59:21 PM",
    "milliseconds_since_epoch": 1528120761743,
    "date": "06-04-2018"
}
{
    "time": "01:59:23 PM",
    "milliseconds_since_epoch": 1528120763647,
    "date": "06-04-2018"
}
```

If we look at the simulation captured, can see that the requests have the `requiresState` fields set. You will see Hoverfly has added a state variable called sequence:1 that acts as a counter.

```
"requiresState": {
    "sequence:1": "1"
}

"requiresState": {
    "sequence:1": "2"
}
```

We can also see that the first response has *transitionsState*` field set.

```
"transitionsState": {
    "sequence:1": "2"
}
```

Note that Hoverfly will automatically set the state for any "sequence:" key to "1" on import. If you want to use a more meaningful key name you will need to initialise the state as follows:

```
hoverctl state set shopping-basket empty
```

See also:

For a more detailed explaination of how sequences work in hoverfly: see *Sequences* in the *Key Concepts* section.

Adding Post Serve Action to particular response

See also:

Please carefully read through *Post Serve Action* alongside this tutorial to gain a high-level understanding of what we are about to cover.

PostServeAction allows you to execute custom code after a response has been served in simulate or spy mode.

In this tutorial, we will help you to setup post serve action and execute it after particular response is served.

Let's begin by writing our post serve action. The following python script makes an HTTP call to http://ip.jsontest.com and prints out your IP address.

We can make a call to any other URL as well in order to send webhook or initiate some processing once response is served. Although the example script is written in python, you can also write an action using any other language as long as you have provided a binary in your local environment to execute the code.

Save the following as `post_serve_action.py`:

```python
#!/usr/bin/env python
import sys
import requests


def get_ip_info():
    url = "http://ip.jsontest.com/"

    try:
        response = requests.get(url)
        response.raise_for_status()  # Raise an exception if the request was not
→successful (HTTP status code >= 400)

        data = response.json()  # Parse the JSON response
        return data
    except requests.exceptions.RequestException as e:
        print(f"An error occurred: {e}")
        return None

def main():

    data = sys.stdin.readlines()
    # this is a request-response pair json string. We can use this pair to make outbound
→call or do any operations
    payload = data[0]
    ip_info = get_ip_info()
    if ip_info:
            print(f"HTTP call invoked from IP Address: {ip_info['ip']}")
    else:
            print("Failed to retrieve IP information.")

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

Start Hoverfly and register a post serve action:

```
hoverctl start
hoverctl post-serve-action set --binary python3 --name callback-script --script <path to␣
↪script directory>/post_serve_action.py --delay 3000
```

Once the post serve action is registered, you can confirm using below hoverctl command.

```
hoverctl post-serve-action get-all

Sample output
+-----------------+---------+--------------------------------+-----------+
|   ACTION NAME   | BINARY  |             SCRIPT             | DELAY(MS) |
+-----------------+---------+--------------------------------+-----------+
| callback-script | python3 | #!/usr/bin/env python import    |      3000 |
|                 |         | sys import logging import       |           |
|                 |         | random from time import sleep   |           |
|                 |         | ...                             |           |
+-----------------+---------+--------------------------------+-----------+
```

Copy this simulation JSON content to a file called `simulation.json`:

```
{
  "data": {
    "pairs": [
      {
        "request": {
          "path": [
            {
              "matcher": "exact",
              "value": "/"
            }
          ],
          "method": [
            {
              "matcher": "exact",
              "value": "GET"
            }
          ],
          "destination": [
            {
              "matcher": "exact",
              "value": "date.jsontest.com"
            }
          ],
          "scheme": [
            {
              "matcher": "exact",
              "value": "http"
            }
          ],
          "body": [
            {
              "matcher": "exact",
              "value": ""
```

(continues on next page)

```
          }
        ]
      },
      "response": {
        "status": 200,
        "body": "01-01-1111",
        "encodedBody": false,
        "templated": false,
        "postServeAction": "callback-script"
      }
    }
  ]
},
"meta": {
  "schemaVersion": "v5.2",
  "hoverflyVersion": "v1.5.3",
  "timeExported": "2023-09-04T11:50:40+05:30"
}
}
```

Run this hoverctl command to import the simulation file.

```
hoverctl import <path-to-simulation-file>
```

The simulation sets hoverfly to return a successful response and 3 seconds after that invokes the "callback-script" action. You can try it out by making the following request to http://date.jsontest.com using cURL.

```
curl --proxy http://localhost:8500 http://date.jsontest.com
```

You should see the message in the hoverfly logs - *Output from post serve action HTTP call invoked from IP Address*.

```
hoverctl logs
```

2.4.2 Advanced tutorials

Using Hoverfly behind a proxy

In some environments, you may only be able to access the internet via a proxy. For example, your organization may route all traffic through a proxy for security reasons.

If this is the case, you will need to configure Hoverfly to work with the 'upstream' proxy.

This configuration value can be easily set when starting an instance of Hoverfly.

For example, if the 'upstream' proxy is running on port `8080` on host `corp.proxy`:

```
hoverctl start --upstream-proxy http://corp.proxy:8080
```

Configuring Hoverfly with a PAC file

If you have a PAC file for configuring clients with the proxy set up in your environment, you can use this to configure Hoverfly.

For example:

```
hoverctl start --pac-file path/to/file.pac
```

Upstream proxy authentication

If the proxy you are using uses HTTP basic authentication, you can provide the authentication credentials as part of the upstream proxy configuration setting.

For example:

```
hoverctl start --upstream-proxy http://my-user:my-pass@corp.proxy:8080
```

Currently, HTTP basic authentication is the only supported authentication method for an authenticated proxy.

Controlling a remote Hoverfly instance with hoverctl

So far, the tutorials have shown how hoverctl can be used to control an instance of Hoverfly running on the same machine.

In some cases, you may wish to use hoverctl to control an instance of Hoverfly running on a remote host. With hoverctl, you can do this using the **targets** feature.

In this example, we assume that the remote host is reachable at `hoverfly.example.com`, and that ports `8880` and `8555` are available. We will also assume that the Hoverfly binary is installed on the remote host.

On the **remote host**, start Hoverfly using flags to override the default admin port (`-ap`) and proxy port (`-pp`).

```
hoverfly -ap 8880 -pp 8555
```

See also:

For a full list of all Hoverfly flags, please refer to *Hoverfly commands* in the *Reference* section.

On your **local machine**, you can create a **target** named `remote` using hoverctl. This target will be configured to communicate with Hoverfly.

```
hoverctl targets create remote \
    --host hoverfly.example.com \
    --admin-port 8880 \
    --proxy-port 8555
```

Now that hoverctl knows the location of the `remote` Hoverfly instance, run the following commands **on your local machine** to capture and simulate a URL using this instance:

```
hoverctl -t remote mode capture
curl --proxy http://hoverfly.example.com:8555 http://ip.jsontest.com
hoverctl -t remote mode simulate
curl --proxy http://hoverfly.example.com:8555 http://ip.jsontest.com
```

You will now need to specify the `remote` target every time you want to interact with this Hoverfly instance. If you are only working with this remote instance, you can set it to be the default target instance for hoverctl.

```
hoverctl targets default remote
```

Note: The `--host` value of the hoverctl target allows hoverctl to interact with the **admin API** of the remote Hoverfly instance.

The application that is making the request (in this case, cURL), **also** needs to be configured to use the remote Hoverfly instance as a proxy. In this example, it is done using cURL's `--proxy` flag.

If you are running Hoverfly on a remote host, you may wish to enable authentication on the Hoverfly proxy and admin API. This is described in the *Enabling authentication for the Hoverfly proxy and API* tutorial.

Enabling authentication for the Hoverfly proxy and API

Sometimes, you may need authentication on Hoverfly. An example use case would be when running Hoverfly on a remote host.

Hoverfly can provide authentication for both the admin API (using a JWT token) and the proxy (using HTTP basic auth).

Setting Hoverfly authentication credentials

To start a Hoverfly instance with authentication enabled, you need to run the `hoverctl start` command with the authentication (`--auth`) flag.

```
hoverctl start --auth
```

Running this command will prompt you to set a username and password for the Hoverfly instance.

This can be bypassed by providing the `--username` and `--password` flags, although **this will leave credentials in your terminal history**.

Warning: By default, hoverctl will start Hoverfly with authentication disabled. If you require authentication you must make sure the `--auth` flag are supplied every time Hoverfly is started.

Logging in to a Hoverfly instance with hoverctl

Now that a Hoverfly instance has started with authentication enabled, you will need to **login** to the instance using hoverctl.

```
hoverctl login
```

Running this command will prompt you to enter the username and password you set for the Hoverfly instance. Again, this can be bypassed by providing the `--username` and `--password` flags.

There may be situations in which you need to log into to a Hoverfly instance that is already running. In this case, it is best practice to create a new **target** for the instance (please see *Controlling a remote Hoverfly instance with hoverctl* for more information on **targets**). You can do this using the `--new-target` flag.

In this example, a remote Hoverfly instance is already running on the host `hoverfly.example.com`, with the ports set to 8880 and 8555 and authentication enabled (the example from *Controlling a remote Hoverfly instance with hoverctl*). You will need to create a new target (named `remote`) for the instance and log in with it.

```
hoverctl login --new-target remote \
    --host hoverfly.example.io \
    --admin-port 8880 \
    --proxy-port 8550
```

You will be prompted to enter the username and password for the instance.

Now run the following commands to capture and simulate a URL using the remote Hoverfly:

```
hoverctl -t remote mode capture
curl --proxy http://my-user:my-pass@hoverfly.example.com:8555 http://ip.jsontest.com
hoverctl -t remote mode simulate
curl --proxy http://my-user:my-pass@hoverfly.example.com:8555 http://ip.jsontest.com
```

Configuring SSL in Hoverfly

Hoverfly supports both one-way and two-way SSL authentication.

Hoverfly uses default certificate which you should add to your HTTPS client's trust store for one-way SSL authentication. You have options to provide your own certificate, please see below.

Override default certificate for one-way SSL authentication

In some cases, you may not wish to use Hoverfly's default SSL certificate. Hoverfly allows you to generate a new certificate and key.

The following command will start a Hoverfly process and create new `cert.pem` and `key.pem` files in the current working directory. These newly-created files will be loaded into the running Hoverfly instance.

```
hoverfly -generate-ca-cert
```

Optionally, you can provide a custom certificate name and authority:

```
hoverfly -generate-ca-cert -cert-name tutorial.cert -cert-org "Tutorial Certificate␣
→Authority"
```

Once you have generated `cert.pem` and `key.pem` files with Hoverfly, you can use hoverctl to start an instance of Hoverfly using these files.

```
hoverctl start --certificate cert.pem --key key.pem
```

Note: Both a certificate and a key file must be supplied. The files must be in unencrypted PEM format.

Configure Hoverfly for two-way SSL authentication

For two-way or mutual SSL authentication, you should provide Hoverfly with a client certificate and a certificate key that you use to authenticate with the remote server.

Two-way SSL authentication is only enabled for request hosts that match the value you provided to the `--client-authentication-destination` flag. You can also pass a regex pattern if you need to match multiple hosts.

```
hoverctl start --client-authentication-client-cert cert.pem --client-authentication-
↪client-key key.pem --client-authentication-destination <host name of the remote server>
```

If you need to provide a CA cert, you can do so using the `--client-authentication-ca-cert` flag.

2.5 Troubleshooting

2.5.1 Why isn't Hoverfly matching my request?

When Hoverfly cannot match a response to an incoming request, it will return information on the closest match:

```
Hoverfly Error!

There was an error when matching

Got error: Could not find a match for request, create or record a valid matcher first!

The following request was made, but was not matched by Hoverfly:

{
    "Path": "/closest-miss",
    "Method": "GET",
    "Destination": "destination.com",
    "Scheme": "http",
    "Query": "",
    "Body": "",
    "Headers": {
        "Accept-Encoding": [
            "gzip"
        ],
        "User-Agent": [
            "Go-http-client/1.1"
        ]
    }
}

The matcher which came closest was:

{
    "path": [
        {
            "matcher": "exact",
            "value": "/closest-miss"
```

(continues on next page)

```
        }
    ],
    "destination": [
        {
            "matcher": "exact",
            "value": "destination.com"
        }
    ],
    "body": [
        {
            "matcher": "exact",
            "value": "body"
        }
    ]
}

But it did not match on the following fields:

[body]

Which if hit would have given the following response:

{
    "status": 200,
    "body": "",
    "encodedBody": false,
    "headers": null
}`
```

Here, you can see which fields did not match. In this case, it was the body. You can also view this information by running `hoverctl logs`.

2.5.2 Why isn't Hoverfly returning the closest match when it cannot match a request?

Hoverfly will only provide this information when the matching strategy is set to **strongest match** (the default). If you are using the **first match** matching strategy, the closet match information will not be returned.

2.5.3 How can I view the Hoverfly logs?

```
hoverctl logs
```

2.5.4 Why does my simulation have a `deprecatedQuery` field?

Older simulations that have been upgraded through newer versions of Hoverfly may now contain a field on requests called `deprecatedQuery`. With the v5 simulation schema, the request query field was updated to more fully represent request query paramters. This involves storing queries based on query keys, similarly to how headers are stored in a simulation.

Currently the `deprecatedQuery` field will work and works alongside the `query` field and support for this field will eventually be dropped.

If you have `deprecatedQuery` field, you should remove it by splitting it by query keys.

```
"deprecatedQuery": "page=20&pageSize=15"
```

```
"query": {
    "page": [
        {
            "matcher": "exact",
            "value": "20"
        }
    ],
    "pageSize": [
        {
            "matcher": "exact",
            "value": "15"
        }
    ],
}
```

If you cannot update your `deprecatedQuery` from your simulation for a technical reason, feel free to raise an issue on Hoverfly.

2.5.5 Why am I not able to access my Hoverfly remotely?

That's because Hoverfly is bind to loopback interface by default, meaning that you can only access to it on localhost. To access it remotely, you can specify the IP address it listens on. For example, setting `0.0.0.0` to listen on all network interfaces.

```
hoverfly -listen-on-host 0.0.0.0
```

2.5.6 My simulation file is very large because of response bodies, what can I do with that?

You can move those response bodies into separate files and specify `bodyFile` in the response instead of `body`. Please refer to *Request Responses Pairs*.

2.6 Reference

These reference documents contain information regarding invoking the hoverctl command, hoverfly command, and interacting with the APIs.

2.6.1 hoverctl commands

This page contains the output of:

```
hoverctl --help
```

The command's help content has been placed here for convenience.

```
hoverctl is the command line tool for Hoverfly

Usage:
  hoverctl [command]

Available Commands:
  completion            Create Bash completion file for hoverctl
  config                Show hoverctl configuration information
  delete                Delete Hoverfly simulation
  destination           Get and set Hoverfly destination
  diff                  Manage the diffs for Hoverfly
  export                Export a simulation from Hoverfly
  flush                 Flush the internal cache in Hoverfly
  import                Import a simulation into Hoverfly
  login                 Login to Hoverfly
  logs                  Get the logs from Hoverfly
  middleware            Get and set Hoverfly middleware
  mode                  Get and set the Hoverfly mode
  post-serve-action     Manage the post-serve-action for Hoverfly
  simulation            Manage the simulation for Hoverfly
  start                 Start Hoverfly
  state                 Manage the state for Hoverfly
  status                Get the current status of Hoverfly
  stop                  Stop Hoverfly
  targets               Get the current targets registered with hoverctl
  templating-data-source Manage the templating data source for Hoverfly
  version               Get the version of hoverctl

Flags:
  -f, --force           Bypass any confirmation when using hoverctl
  -h, --help            help for hoverctl
      --set-default     Sets the current target as the default target for hoverctl
```

```
 -t, --target string    A name for an instance of Hoverfly you are trying to communicate␣
→with. Overrides the default target (default)
 -v, --verbose          Verbose logging from hoverctl

Use "hoverctl [command] --help" for more information about a command.
```

hoverctl auto completion

hoverctl supplies auto completion for Bash. Run the following command to install the completions.

```
hoverctl completion
```

This will create the completion file in your hoverfly directory and create a symbolic link in your bash_completion.d folder.

Optionally you can supply a location for the symbolic link as an argument to the completion command.

```
hoverctl completion /usr/local/etc/bash_completion.d/hoverctl
```

2.6.2 Hoverfly commands

This page contains the output of:

```
hoverfly --help
```

The command's help content has been placed here for convenience.

```
Usage of hoverfly:
  -add
        Add new user '-add -username hfadmin -password hfpass'
  -admin
        Supply '-admin=false' to make this non admin user (default true)
  -ap string
        Admin port - run admin interface on another port (i.e. '-ap 1234' to run admin␣
→UI on port 1234)
  -auth
        Enable authentication
  -cache-size int
        Set the size of request/response cache (default 1000)
  -capture
        Start Hoverfly in capture mode - transparently intercepts and saves requests/
→response
  -cert string
        CA certificate used to sign MITM certificates
  -cert-name string
        Cert name (default "hoverfly.proxy")
  -cert-org string
        Organisation name for new cert (default "Hoverfly Authority")
  -client-authentication-ca-cert string
        Path to the ca cert file used for authentication
  -client-authentication-client-cert string
```

```
        Path to the client certification file used for authentication
 -client-authentication-client-key string
        Path to the client key file used for authentication
 -client-authentication-destination string
        Regular expression of destination with client authentication
 -cors
        Enable CORS support
 -db string
        Storage to use - 'boltdb' or 'memory' which will not write anything to disk␣
→(DEPRECATED) (default "memory")
 -db-path string
        A path to a BoltDB file with persisted user and token data for authentication␣
→(DEPRECATED)
 -dest value
        Specify which hosts to process (i.e. '-dest fooservice.org -dest barservice.org -
→dest catservice.org') - other hosts will be ignored will passthrough'
 -destination string
        Control which URLs Hoverfly should intercept and process, it can be string or␣
→regex (default ".")
 -dev
        Enable CORS headers to allow Hoverfly Admin UI development
 -dev-cors-origin string
        Custom CORS origin for dev mode (default "http://localhost:4200")
 -diff
        Start Hoverfly in diff mode - calls real server and compares the actual response␣
→with the expected simulation config if present
 -disable-cache
        Disable the request/response cache (the cache that sits in front of matching)
 -generate-ca-cert
        Generate CA certificate and private key for MITM
 -import value
        Import from file or from URL (i.e. '-import my_service.json' or '-import http://
→mypage.com/service_x.json'
 -journal-size int
        Set the size of request/response journal (default 1000)
 -journal-indexing-key string
        Specify the index key using which you want to index journal. Index shares same␣
→syntax as the one for templating, such as Request.QueryParam.myParam or Request.Header.
→X-Header-Id.[1].
        It is used for extracting the data from the journal entry to use as a key for␣
→that entry.
 -key string
        Private key of the CA used to sign MITM certificates
 -listen-on-host string
        Specify which network interface to bind to, eg. 0.0.0.0 will bind to all␣
→interfaces. By default hoverfly will only bind ports to loopback interface
 -log-level string
        Set log level (panic, fatal, error, warn, info or debug) (default "info")
 -log-no-color
        Disable colors for logging
 -log-no-quotes
        Disable quoting and escaping of logged fields
```

```
 -logs string
       Specify format for logs, options are "plaintext" and "json" (default "plaintext")
 -logs-file string
       Specify log file name for output logs (default "hoverfly.log")
 -logs-output value
       Specify locations for output logs, options are "console" and "file" (default
↪"console")
 -logs-size int
       Set the amount of logs to be stored in memory (default 1000)
 -metrics
       Enable metrics logging to stdout
 -post-serve-action string
       Set local post serve action by passing the action name, binary and the path of␣
↪the action script and delay in Ms separated by space.
       (i.e. -post-serve-action "<action name> python3 <script path to load> 1000" -
↪post-serve-action "<action name> python3 <script path to load> 3000")
       Set remote post serve action by passing the action name, remote host and delay␣
↪in Ms separated by space.
        (i.e. -post-serve-action "<action name> <http/https remote host> 1000" -post-
↪serve-action "<action name> <http/https remote host> 3000")
       We can set multiple post serve actions and use in our simulation schema file.
 -middleware string
       Set middleware by passing the name of the binary and the path of the middleware␣
↪script separated by space. (i.e. '-middleware "python script.py"')
 -modify
       Start Hoverfly in modify mode - applies middleware (required) to both outgoing␣
↪and incoming HTTP traffic
 -no-import-check
       Skip duplicate request check when importing simulations
 -password string
       Password for new user
 -password-hash string
       Password hash for new user instead of password
 -plain-http-tunneling
       Use plain http tunneling to host with non-443 port
 -pp string
       Proxy port - run proxy on another port (i.e. '-pp 9999' to run proxy on port␣
↪9999)
 -response-body-files-allow-origin value
       When a response contains a url in bodyFile, it will be loaded only if the origin␣
↪is allowed
 -response-body-files-path string
       When a response contains a relative bodyFile, it will be resolved against this␣
↪path (default is CWD)
 -spy
       Start Hoverfly in spy mode, similar to simulate but calls real server when cache␣
↪miss
 -synthesize
       Start Hoverfly in synthesize mode (middleware is required)
 -templating-data-source
       Set templating CSV data source by passing data source name and CSV data file␣
↪path separated by space.
```

```
        (i.e. -templating-data-source "<data source name> <file path>")
        We can set multiple template CSV data source and then we can query data by using␣
→templating csv helper method
  -tls-verification
        Turn on/off tls verification for outgoing requests (will not try to verify␣
→certificates) (default true)
  -upstream-proxy string
        Specify an upstream proxy for hoverfly to route traffic through
  -username string
        Username for new user
  -v    Should every proxy request be logged to stdout
  -version
        Get the version of hoverfly
  -webserver
        Start Hoverfly in webserver mode (simulate mode)
```

2.6.3 Request matchers

A Request Matcher is used to define the desired value for a specific request field when matching against incoming requests. Given a **matcher value** and **string to match**, each matcher will transform and compare the values in a different way.

Exact matcher

Evaluates the equality of the matcher value and the string to match. There are no transformations. This is the default Request Matcher type which is set by Hoverfly when requests and responses are captured.

Example

```
"matcher": "exact"
"value": "?"
```

Negation matcher

The opposite of the Exact matcher. This matcher will evaluate to true if the values being compared are not the same.

Example

```
"matcher": "negate"
"value": "?"
```

Glob matcher

Allows wildcard matching (similar to BASH) using the ***** character.

Example

```
"matcher": "glob"
"value": "?"
```

Regex matcher

Parses the matcher value as a regular expression which is then executed against the string to match. This will pass only if the regular expression successfully returns a result.

Example

```
"matcher": "regex"
"value": "?"
```

XML matcher

Transforms both the matcher value and string to match into XML objects and then evaluates their equality.

Example

```
"matcher": "xml"
"value": "?"
```

XPath matcher

Parses the matcher value as an XPath expression, transforms the string to match into an XML object and then executes the expression against it. This will pass only if the expression successfully returns a result.

Example

```
"matcher": "xpath"
"value": "?"
```

JSON matcher

Transforms both the matcher value and string to match into JSON objects and then evaluates their equality.

Example

```
"matcher": "json"
"value": "?"
```

JSON partial matcher

Unlike a JSON matcher which does the full matching of two JSON documents, this matcher evaluates if the matcher value is a subset of the incoming JSON document. The matcher ignores any absent fields and lets you match only the part of JSON document you care about.

Example

```
"matcher": "jsonPartial"
"value": "?"
```

JSONPath matcher

Parses the matcher value as a JSONPath expression, transforms the string to match into a JSON object and then executes the expression against it. This will pass only if the expression successfully returns a result.

Example

```
"matcher": "jsonpath"
"value": "?"
```

Form matcher

Matches form data posted in the request payload with content type `application/x-www-form-urlencoded`. You can match only the form params you are interested in regardless of the order. You can also leverage `jwt` or `jsonpath` matchers if your form params contains JWT tokens or JSON document.

Please note that this matcher only works for `body` field.

Example

```
"matcher": "form",
"value": {
    "grant_type": [
      {
        "matcher": "exact",
        "value": "authorization_code"
      }
    ],
  "client_assertion": [
    {
      "matcher": "jwt",
      "value": "{\"header\":{\"alg\":\"HS256\"},\"payload\":{\"sub\":\"1234567890\",\
→"name\":\"John Doe\"}}"
    }
  ]
}
```

Array matcher

Matches an array contains exactly the given values and nothing else. This can be used to match multi-value query param or header in the request data.

The following configuration options are available to change the behaviour of the matcher:

- ignoreOrder - ignore the order of the values.

- ignoreUnknown - ignore any extra values.

- ignoreOccurrences - ignore any duplicated values.

Example

```
"matcher": "array",
"config": {
    "ignoreUnknown": "<true/false>",
    "ignoreOrder": "<true/false>",
    "ignoreOccurrences": "<true/false>"
},
"value": [
    "access:vod",
    "order:latest",
    "profile:vd"
]
```

JWT matcher

This matcher is primarily used for matching JWT tokens. This matcher converts base64 encoded JWT to JSON document `{"header": {}, "payload": ""}` and does JSON partial match with the matcher value.

Matcher value contains only keys that they want to match in JWT.

Example

```
"matcher": "jwt"
"value": "{\"header\":{\"alg\":\"HS256\"},\"payload\":{\"sub\":\"1234567890\",\"name\":\
→"John Doe\"}}"
```

Matcher chaining

Matcher chaining allows you to pass a matched value into another matcher to do further matching.

It typically removes the stress of composing and testing complex expressions and make matchers more readable.

For example, one can use JSONPath to get a JSON node, then use another matcher to match the JSON node value as follows.

Example

```
"matcher": "jsonpath",
"value": "$.user.id",
"doMatch": {
    "matcher": "exact",
    "value": "1"
}
```

2.6.4 REST API

GET /api/v2/simulation

Gets all simulation data. The simulation JSON contains all the information Hoverfly can hold; this includes recordings, templates, delays and metadata.

Example response body

```
{
  "data": {
    "pairs": [
      {
        "request": {
          "path": [
            {
              "matcher": "exact",
              "value": "/"
            }
          ],
          "method": [
            {
              "matcher": "exact",
              "value": "GET"
            }
          ],
          "destination": [
            {
              "matcher": "exact",
              "value": "myhost.io"
            }
          ],
          "scheme": [
            {
              "matcher": "exact",
              "value": "https"
            }
          ],
          "body": [
            {
              "matcher": "exact",
              "value": ""
            }
```

```
        ],
        "headers": {
          "Accept": [
            {
              "matcher": "glob",
              "value": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
→webp,*/*;q=0.8"
            }
          ],
          "Content-Type": [
            {
              "matcher": "glob",
              "value": "text/plain; charset=utf-8"
            }
          ],
          "User-Agent": [
            {
              "matcher": "glob",
              "value": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/
→537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"
            }
          ]
        },
        "query": {
          "status": [
            {
              "matcher": "exact",
              "value": "available"
            }
          ]
        }
      },
      "response": {
        "status": 200,
        "body": "<h1>Matched on recording</h1>",
        "encodedBody": false,
        "headers": {
          "Content-Type": [
            "text/html; charset=utf-8"
          ]
        },
        "templated": false
      }
    },
    {
      "request": {
        "path": [
          {
            "matcher": "exact",
            "value": "/template"
          }
        ]
```

```
      },
      "response": {
        "status": 200,
        "body": "<h1>Matched on template</h1>",
        "encodedBody": false,
        "headers": {
          "Content-Type": [
            "text/html; charset=utf-8"
          ]
        },
        "templated": false
      }
    }
  ],
  "globalActions": {
    "delays": [],
    "delaysLogNormal": []
  }
  },
  "meta": {
    "schemaVersion": "v5",
    "hoverflyVersion": "v1.0.0",
    "timeExported": "2019-05-30T22:14:24+01:00"
  }
}
```

PUT /api/v2/simulation

This puts the supplied simulation JSON into Hoverfly, overwriting any existing simulation data.

Example request body

```
{
  "data": {
    "pairs": [
      {
        "request": {
          "path": [
            {
              "matcher": "exact",
              "value": "/"
            }
          ],
          "method": [
            {
              "matcher": "exact",
              "value": "GET"
            }
          ],
          "destination": [
            {
```

```
            "matcher": "exact",
            "value": "myhost.io"
          }
        ],
        "scheme": [
          {
            "matcher": "exact",
            "value": "https"
          }
        ],
        "body": [
          {
            "matcher": "exact",
            "value": ""
          }
        ],
        "headers": {
          "Accept": [
            {
              "matcher": "glob",
              "value": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
→webp,*/*;q=0.8"
            }
          ],
          "Content-Type": [
            {
              "matcher": "glob",
              "value": "text/plain; charset=utf-8"
            }
          ],
          "User-Agent": [
            {
              "matcher": "glob",
              "value": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/
→537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"
            }
          ]
        },
        "query": {
          "status": [
            {
              "matcher": "exact",
              "value": "available"
            }
          ]
        }
      },
      "response": {
        "status": 200,
        "body": "<h1>Matched on recording</h1>",
        "encodedBody": false,
        "headers": {
```

```
          "Content-Type": [
            "text/html; charset=utf-8"
          ]
        },
        "templated": false
      }
    },
    {
      "request": {
        "path": [
          {
            "matcher": "exact",
            "value": "/template"
          }
        ]
      },
      "response": {
        "status": 200,
        "body": "<h1>Matched on template</h1>",
        "encodedBody": false,
        "headers": {
          "Content-Type": [
            "text/html; charset=utf-8"
          ]
        },
        "templated": false
      }
    }
  ],
  "globalActions": {
    "delays": [],
    "delaysLogNormal": []
  }
},
"meta": {
  "schemaVersion": "v5",
  "hoverflyVersion": "v1.0.0",
  "timeExported": "2019-05-30T22:14:24+01:00"
}
}
```

POST /api/v2/simulation

This appends the supplied simulation JSON to the existing simulation data in Hoverfly. Any pair that has request data identical to the existing ones will not be added.

Example request body

```
{
  "data": {
    "pairs": [
```

Chapter 2. Contents

```
    {
      "request": {
        "path": [
          {
            "matcher": "exact",
            "value": "/"
          }
        ],
        "method": [
          {
            "matcher": "exact",
            "value": "GET"
          }
        ],
        "destination": [
          {
            "matcher": "exact",
            "value": "myhost.io"
          }
        ],
        "scheme": [
          {
            "matcher": "exact",
            "value": "https"
          }
        ],
        "body": [
          {
            "matcher": "exact",
            "value": ""
          }
        ],
        "headers": {
          "Accept": [
            {
              "matcher": "glob",
              "value": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
→webp,*/*;q=0.8"
            }
          ],
          "Content-Type": [
            {
              "matcher": "glob",
              "value": "text/plain; charset=utf-8"
            }
          ],
          "User-Agent": [
            {
              "matcher": "glob",
              "value": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/
→537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"
            }
```

```
            ]
          },
          "query": {
            "status": [
              {
                "matcher": "exact",
                "value": "available"
              }
            ]
          }
        },
        "response": {
          "status": 200,
          "body": "<h1>Matched on recording</h1>",
          "encodedBody": false,
          "headers": {
            "Content-Type": [
              "text/html; charset=utf-8"
            ]
          },
          "templated": false
        }
      },
      {
        "request": {
          "path": [
            {
              "matcher": "exact",
              "value": "/template"
            }
          ]
        },
        "response": {
          "status": 200,
          "body": "<h1>Matched on template</h1>",
          "encodedBody": false,
          "headers": {
            "Content-Type": [
              "text/html; charset=utf-8"
            ]
          },
          "templated": false
        }
      }
    ],
    "globalActions": {
      "delays": [],
      "delaysLogNormal": []
    }
  },
  "meta": {
    "schemaVersion": "v5",
```

```
        "hoverflyVersion": "v1.0.0",
        "timeExported": "2019-05-30T22:14:24+01:00"
    }
}
```

DELETE /api/v2/simulation

Unsets the simulation data for Hoverfly.

GET /api/v2/simulation/schema

Gets the JSON Schema used to validate the simulation JSON.

GET /api/v2/hoverfly

Gets configuration information from the running instance of Hoverfly.

Example response body

```
{
    "cors": {
        "enabled": true,
        "allowOrigin": "*",
        "allowMethods": "GET,POST,PUT,PATCH,DELETE,HEAD,OPTIONS",
        "allowHeaders": "Content-Type,Origin,Accept,Authorization,Content-Length,X-
→Requested-With",
        "preflightMaxAge": 1800,
        "allowCredentials": true
    },
    "destination": ".",
    "middleware": {
        "binary": "python",
        "script": "# a python script would go here",
        "remote": ""
    },
    "mode": "simulate",
    "arguments": {
        "matchingStrategy": "strongest"
    },
    "isWebServer": false,
    "usage": {
        "counters": {
            "capture": 0,
            "modify": 0,
            "simulate": 0,
            "synthesize": 0
        }
```

```
    },
    "version": "v1.3.3",
    "upstreamProxy": ""
}
```

GET /api/v2/hoverfly/cors

Gets CORS configuration information from the running instance of Hoverfly.

Example response body

```
{
    "enabled": true,
    "allowOrigin": "*",
    "allowMethods": "GET,POST,PUT,PATCH,DELETE,HEAD,OPTIONS",
    "allowHeaders": "Content-Type, Origin, Accept, Authorization, Content-Length, X-
→Requested-With",
    "preflightMaxAge": 1800,
    "allowCredentials": true
}
```

GET /api/v2/hoverfly/destination

Gets the current destination setting for the running instance of Hoverfly.

Example response body

```
{
    destination: "."
}
```

PUT /api/v2/hoverfly/destination

Sets a new destination for the running instance of Hoverfly, overwriting the existing destination setting.

Example request body

```
{
    destination: "new-destination"
}
```

GET /api/v2/hoverfly/middleware

Gets the middleware settings for the running instance of Hoverfly. This could be either an executable binary, a script that can be executed with a binary or a URL to remote middleware.

Example response body

```
{
    "binary": "python",
    "script": "#python code goes here",
    "remote": ""
}
```

PUT /api/v2/hoverfly/middleware

Sets new middleware, overwriting the existing middleware for the running instance of Hoverfly. The middleware being set can be either an executable binary located on the host, a script and the binary to execute it or the URL to a remote middleware.

Example request body

```
{
    "binary": "python",
    "script": "#python code goes here",
    "remote": ""
}
```

GET /api/v2/hoverfly/post-serve-action

Get all the post serve actions for the running instance of Hoverfly. It will return list of scripts that can be executed with a binary after response is served back.

Example response body

```
{
    "actions": [
        {
            "actionName": "<post serve action name>",
            "binary": "python",
            "script": "#python code goes here",
            "delayInMs":"#delay(in ms) post which script will be executed after serving␣
→the request",
            "remote": "<url of the remote action webserver if set>"
        }
    ]
}
```

PUT /api/v2/hoverfly/post-serve-action

Sets new post serve action, overwriting the existing post serve action for the running instance of Hoverfly. The post serve action being set is an executable binary located on the host. We can set multiple post serve actions. It returns all the post serve actions.

Example request body

```
{
    "actionName": "#post serve action names goes here",
    "binary": "python",
    "script": "#python code goes here",
    "delayInMs": "#delay(in ms) post which script will be executed after serving the␣
→request"
    "remote": "<url of the remote action webserver>"
}
```

DELETE /api/v2/hoverfly/post-serve-action/:actionName

Delete a particular post serve action for the running instance of Hoverfly. It returns all the remaining post serve actions.

GET /api/v2/hoverfly/mode

Gets the mode for the running instance of Hoverfly.

Example response body

```
{
    "mode": "capture",
    "arguments": {
        "headersWhitelist": [
            "*"
        ],
        "stateful": true,
        "overwriteDuplicate": true
    }
}
```

PUT /api/v2/hoverfly/mode

Changes the mode of the running instance of Hoverfly. Pass additional arguments to set the mode options.

Example request body

```
{
    "mode": "capture",
    "arguments": {
        "headersWhitelist": [
            "*"
```

```
        ],
        "stateful": true,
        "overwriteDuplicate": true
    }
}
```

GET /api/v2/hoverfly/usage

Gets metrics information for the running instance of Hoverfly.

Example response body

```
{
    "metrics": {
        "counters": {
            "capture": 0,
            "modify": 0,
            "simulate": 0,
            "synthesize": 0
        }
    }
}
```

GET /api/v2/hoverfly/version

Gets the version of Hoverfly.

Example response body

```
{
    "version": "v0.10.1"
}
```

GET /api/v2/hoverfly/upstream-proxy

Gets the upstream proxy configured for Hoverfly.

Example response body

```
{
    "upstreamProxy": "proxy.corp.big-it-company.org:8080"
}
```

GET /api/v2/hoverfly/pac

Gets the PAC file configured for Hoverfly. The response contains plain text with PAC file. If no PAC was provided before, the response is 404 with contents:

```
{
    "error": "Not found"
}
```

PUT /api/v2/hoverfly/pac

Sets the PAC file for Hoverfly.

DELETE /api/v2/hoverfly/pac

Unsets the PAC file configured for Hoverfly.

GET /api/v2/cache

Gets the requests and responses stored in the cache.

Example response body

```
{
    "cache": [
        {
            "key": "2fc8afceec1b6bcf99ff1f547c1f5b11",
            "matchingPair": {
                "request": {
                    "path": [{
                        "matcher": "exact",
                        "value": "hoverfly.io"
                    }]
                },
                "response": {
                    "status": 200,
                    "body": "response body",
                    "encodedBody": false,
                    "headers": {
                        "Hoverfly": [
                            "Was-Here"
                        ]
                    }
                }
            },
            "headerMatch": false,
```

(continues on next page)

```
            "closestMiss": null
        }
    ]
}
```

DELETE /api/v2/cache

Delete all requests and responses stored in the cache.

GET /api/v2/logs

Gets the logs from Hoverfly.

It supports multiple parameters to limit the amount of entries returned:

- `limit` - Maximum amount of entries. 500 by default;
- `from` - Timestamp to start filtering from.

Running hoverfly with `-logs-size=0` disables logging and 500 response is returned with body:

```
{
    "error": "Logs disabled"
}
```

Example response body

```
{
    "logs": [
        {
            "level": "info",
            "msg": "serving proxy",
            "time": "2017-03-13T12:22:39Z"
        },
        {
            "destination": ".",
            "level": "info",
            "mode": "simulate",
            "msg": "current proxy configuration",
            "port": "8500",
            "time": "2017-03-13T12:22:39Z"
        },
        {
            "destination": ".",
            "Mode": "simulate",
            "ProxyPort": "8500",
            "level": "info",
            "msg": "Proxy prepared...",
            "time": "2017-03-13T12:22:39Z"
```

```
        },
    ]
}
```

GET /api/v2/journal

Gets the journal from Hoverfly. Each journal entry contains both the request Hoverfly received and the response it served along with the mode Hoverfly was in, the time the request was received and the time taken for Hoverfly to process the request. Latency is in milliseconds. It also returns its corresponding indexes.

It supports paging using the `offset` and `limit` query parameters.

It supports multiple parameters to limit the amount of entries returned:

- `limit` - Maximum amount of entries. 500 by default;

- `offset` - Offset of the first element;

- `to` - Timestamp to start filtering to;

- `from` - Timestamp to start filtering from;

- `sort` - Sort results in format "field:order". Supported fields: `timestarted` and `latency`. Supported orders: `asc` and `desc`.

It also returns post serve action details containing action name, when it was invoked, completed, correlation ID and HTTP status.

Running hoverfly with `-journal-size=0` disables logging and 500 response is returned with body:

```
{
    "error": "Journal disabled"
}
```

Example response body

```
{
  "journal": [
    {
      "request": {
        "path": "/",
        "method": "GET",
        "destination": "hoverfly.io",
        "scheme": "http",
        "query": "",
        "body": "",
        "headers": {
          "Accept": [
            "*/*"
          ],
          "Proxy-Connection": [
            "Keep-Alive"
          ],
          "User-Agent": [
```

```
            "curl/7.50.2"
          ]
        }
    },
    "response": {
      "status": 502,
      "body": "Hoverfly Error!\n\nThere was an error when matching\n\nGot error: Could␣
→not find a match for request, create or record a valid matcher first!",
      "encodedBody": false,
      "headers": {
        "Content-Type": [
          "text/plain"
        ]
      }
    },
    "postServeAction": {
        "name": "test-callback",
        "invoked": "2024-04-09T00:56:11.606+00:30",
        "completed": "2024-04-09T00:56:12.619+00:30",
        "correlationId": "9a935ec9-fe65-40d8-9ae6-d9d43dabf679",
        "status": 200
    }
    "id":"mOBdPSIIBbjNqBvpZ8H-",
    "mode": "simulate",
    "timeStarted": "2017-07-17T10:41:59.168+01:00",
    "latency": 0.61334
    }
  ],
  "indexes": [
  {
    "name": "Request.destination",
    "entries": [
      {
        "key": "hoverfly.io",
        "journalEntryId": "mOBdPSIIBbjNqBvpZ8H-"
      }
    ]
  }],
  "offset": 0,
  "limit": 25,
  "total": 1
}
```

DELETE /api/v2/journal

Delete all entries stored in the journal.

POST /api/v2/journal

Filter and search entries stored in the journal.

Example request body

```
{
    "request": {
        "destination": [{
          "matcher": "exact",
          "value": "hoverfly.io"
        }]
    }
}
```

GET /api/v2/journal/index

Gets all the journal indexes from Hoverfly. Each Index contains key, extracted value for that particular key and journal index id to which it is pointing to.

Example response body

```
[
  {
    "name": "Request.QueryParam.id",
    "entries": [
      {
        "key": "100",
        "journalEntryId": "ZCyiQtamEtwi-NNU9RT1"
      },
      {
        "key": "101",
        "journalEntryId": "YFU5dm2uDZ4UStX3ldkX"
      }
    ]
  },
  {
    "name": "Request.QueryParam.name",
    "entries": [
      {
        "key": "Test1",
        "journalEntryId": "ZCyiQtamEtwi-NNU9RT1"
      },
      {
        "key": "Test2",
```

```
            "journalEntryId": "YFU5dm2uDZ4UStX3ldkX"
        }
    ]
    },
]
```

POST /api/v2/journal/index

Allow a user to set journal indexing by specifying index key/name. Index name is "request-query" shares the same syntax as the one for templating, such as Request.QueryParam.myParam or Request.Header.X-Header-Id.[1] It's used for extracting the data from the journal entry to use as a key for that entry. It returns all the journal indexes that have been set. It indexes pre-existing or new journal entries.

Example request body

```
{
    "name":"Request.QueryParam.myParam"
}
```

DELETE /api/v2/journal/index/:index-name

Deletes journal index from hoverfly.

GET /api/v2/state

Gets the state from Hoverfly. State is represented as a set of key value pairs.

Example response body

```
{
  "state": {
    "page_state": "CHECKOUT"
  }
}
```

DELETE /api/v2/state

Deletes all state from Hoverfly.

PUT /api/v2/state

Deletes all state from Hoverfly and then sets the state to match the state in the request body.

Example request body

```
{
  "state": {
    "page_state": "CHECKOUT"
  }
}
```

PATCH /api/v2/state

Updates state in Hoverfly. Will update each state key referenced in the request body.

Example request body

```
{
  "state": {
    "page_state": "CHECKOUT"
  }
}
```

GET /api/v2/diff

Gets all reports containing response differences from Hoverfly. The diffs are represented as lists of strings grouped by the same requests.

Example response body

```
{
  "diff": [{
    "request": {
      "method": "GET",
      "host": "time.jsontest.com",
      "path": "/",
      "query": ""
    },
    "diffReports": [{
      "timestamp": "2018-03-16T17:45:40Z",
      "diffEntries": [{
        "field": "header/X-Cloud-Trace-Context",
        "expected": "[ec6c455330b682c3038ba365ade6652a]",
        "actual": "[043c9bb2eafa1974bc09af654ef15dc3]"
      }, {
        "field": "header/Date",
        "expected": "[Fri, 16 Mar 2018 17:45:34 GMT]",
        "actual": "[Fri, 16 Mar 2018 17:45:41 GMT]"
      }, {
```

```
          "field": "body/time",
          "expected": "05:45:34 PM",
          "actual": "05:45:41 PM"
        }, {
          "field": "body/milliseconds_since_epoch",
          "expected": "1.521222334104e+12",
          "actual": "1.521222341017e+12"
        }]
      }]
    }]
}
```

POST /api/v2/diff

Gets reports containing response differences from Hoverfly filtered on basis of excluded criteria provided(i.e. headers and response keys in jsonpath format to exclude). The diffs are in same format as we receive in GET request.

Example request body

```
{
  "excludedHeaders":["Date"],
  "excludedResponseFields":["$.time"]
}
```

DELETE /api/v2/diff

Deletes all reports containing differences from Hoverfly.

GET /api/v2/hoverfly/templating-data-source/csv

Get all the templating data source for the running instance of Hoverfly. It will return list of data sources that has been uploaded by the user to be queried by the template function in the response.

Example response body

```
{
  "csvDataSources": [
    {
      "name": "student-marks",
      "data": "id,name,marks\n1,Test1,300\n2,Test2,600\n"
    }
  ]
}
```

PUT /api/v2/hoverfly/templating-data-source/csv

Sets new template data source, overwriting the existing template data source for the running instance of Hoverfly. The template CSV data source being set is being queried via template function to generate the response. This API call returns back all the templating data source that have been set.

Example request body

```
{
    "name":"student-marks",
    "data":"id,name,marks\n1,Test1,55\n2,Test2,56\n*,Dummy,ABSENT"
}
```

DELETE /api/v2/hoverfly/templating-data-source/csv/:data-source-name

Delete a particular data source for the running instance of Hoverfly. It returns all the remaining template data sources.

DELETE /api/v2/shutdown

Shuts down the hoverfly instance.

2.6.5 Simulation schema

This is the JSON schema for v5 Hoverfly simulations.

```
{
  "additionalProperties": false,
  "definitions": {
    "delay": {
      "properties": {
        "delay": {
          "type": "integer"
        },
        "httpMethod": {
          "type": "string"
        },
        "urlPattern": {
          "type": "string"
        }
      },
      "type": "object"
    },
    "delay-log-normal": {
      "properties": {
        "httpMethod": {
          "type": "string"
        },
        "max": {
          "type": "integer"
```

```
    },
    "mean": {
      "type": "integer"
    },
    "median": {
      "type": "integer"
    },
    "min": {
      "type": "integer"
    },
    "urlPattern": {
      "type": "string"
    }
  },
  "type": "object"
},
"field-matchers": {
  "properties": {
    "matcher": {
      "type": "string"
    },
    "value": {},
    "config": {
      "properties": {
        "ignoreUnknown": {
          "type": "boolean"
        },
        "ignoreOrder": {
          "type": "boolean"
        },
        "ignoreOccurrences": {
          "type": "boolean"
        }
      },
      "type": "object"
    },
    "doMatch": {
      "$ref": "#/definitions/field-matchers"
    }
  },
  "type": "object"
},
"headers": {
  "additionalProperties": {
    "items": {
      "type": "string"
    },
    "type": "array"
  },
  "type": "object"
},
"literals": {
```

```
    "properties": {
      "name": {
        "type": "string"
      },
      "value": {}
    },
    "required": ["name", "value"],
    "type": "object"
  },
  "meta": {
    "properties": {
      "hoverflyVersion": {
        "type": "string"
      },
      "schemaVersion": {
        "type": "string"
      },
      "timeExported": {
        "type": "string"
      }
    },
    "required": ["schemaVersion"],
    "type": "object"
  },
  "request": {
    "properties": {
      "body": {
        "items": {
          "$ref": "#/definitions/field-matchers"
        },
        "type": "array"
      },
      "destination": {
        "items": {
          "$ref": "#/definitions/field-matchers"
        },
        "type": "array"
      },
      "headers": {
        "$ref": "#/definitions/request-headers"
      },
      "path": {
        "items": {
          "$ref": "#/definitions/field-matchers"
        },
        "type": "array"
      },
      "query": {
        "$ref": "#/definitions/request-queries"
      },
      "requiresState": {
        "patternProperties": {
```

```json
        ".{1,}": {
          "type": "string"
        }
      },
      "type": "object"
    },
    "scheme": {
      "items": {
        "$ref": "#/definitions/field-matchers"
      },
      "type": "array"
    }
  },
  "type": "object"
},
"request-headers": {
  "additionalProperties": {
    "items": {
      "$ref": "#/definitions/field-matchers"
    },
    "type": "array"
  },
  "type": "object"
},
"request-queries": {
  "additionalProperties": {
    "items": {
      "$ref": "#/definitions/field-matchers"
    },
    "type": "array"
  },
  "type": "object"
},
"request-response-pair": {
  "properties": {
    "request": {
      "$ref": "#/definitions/request"
    },
    "response": {
      "$ref": "#/definitions/response"
    }
  },
  "required": ["request", "response"],
  "type": "object"
},
"response": {
  "properties": {
    "body": {
      "type": "string"
    },
    "bodyFile": {
      "type": "string"
```

```
    },
    "encodedBody": {
      "type": "boolean"
    },
    "fixedDelay": {
      "type": "integer"
    },
    "headers": {
      "$ref": "#/definitions/headers"
    },
    "logNormalDelay": {
      "properties": {
        "max": {
          "type": "integer"
        },
        "mean": {
          "type": "integer"
        },
        "median": {
          "type": "integer"
        },
        "min": {
          "type": "integer"
        }
      }
    },
    "postServeAction": {
      "type": "string"
    },
    "removesState": {
      "type": "array"
    },
    "status": {
      "type": "integer"
    },
    "templated": {
      "type": "boolean"
    },
    "transitionsState": {
      "patternProperties": {
        ".{1,}": {
          "type": "string"
        }
      },
      "type": "object"
    }
  },
  "type": "object"
},
"variables": {
  "properties": {
    "name": {
```

Chapter 2. Contents

```json
          "type": "string"
        },
        "function": {
          "type": "string"
        },
        "arguments": {
          "type": "array"
        }
      },
      "required": ["name", "function"],
      "type": "object"
    }
  },
  "description": "Hoverfly simulation schema",
  "properties": {
    "data": {
      "properties": {
        "globalActions": {
          "properties": {
            "delays": {
              "items": {
                "$ref": "#/definitions/delay"
              },
              "type": "array"
            },
            "delaysLogNormal": {
              "items": {
                "$ref": "#/definitions/delay-log-normal"
              },
              "type": "array"
            }
          },
          "type": "object"
        },
        "literals": {
          "items": {
            "$ref": "#/definitions/literals"
          },
          "type": "array"
        },
        "pairs": {
          "items": {
            "$ref": "#/definitions/request-response-pair"
          },
          "type": "array"
        },
        "variables": {
          "items": {
            "$ref": "#/definitions/variables"
          },
          "type": "array"
        }
```

```
    },
      "type": "object"
    },
    "meta": {
      "$ref": "#/definitions/meta"
    }
  },
  "required": ["data", "meta"],
  "type": "object"
}
```

2.7 Contributing

Contributions are welcome! To contribute, please:

1. Fork the repository

2. Create a feature branch on your fork

3. Commit your changes, and create a pull request against Hoverfly's master branch

In your pull request, please include details regarding your change (why you made the change, how to test it etc).

Learn more about the forking workflow here.

2.7.1 Building, running & testing

You will need Go 1.22 . Instructions on how to set up your Go environment can be found here.

```
git clone https://github.com/SpectoLabs/hoverfly.git
# or: git clone https://github.com/<your_username>/hoverfly.git
cd hoverfly
make build
```

Notice the binaries are in the `target` directory.

Finally, to test your build:

```
make test
```

2.8 Community

- Chat on the Hoverfly Gitter channel

- Join the Hoverfly mailing list

- Raise a GitHub Issue

- Get in touch with SpectoLabs on Twitter