Flood - User Guide

Kajetan Rzepecki

September 24, 2013

Contents

1 Introduction

Flood is a load simulator useful for automatic **Comet/PUSH application** stress-testing. It is **asynchronous, event based** and enables you to create JSON encoded **test scenarios of arbitrary complexity** involving **tens of thousands of simulated users**, no Erlang required!

1.1 Use cases

Some of the most common use cases that **Flood** might be helpful in testing are:

- **Massive, real-time, on-line chats**,
- **Publisher-Subscriber channels**,
- **Instant messaging**.

However, Flood is general enough to test *any* event-based Comet application that uses the supported protocols.

1.2 Supported Protocols

Flood currently supports the **Socket.IO** protocol over **WebSocket** and **XHR-polling** transports with emphasis on Socket.IO event based communication. Flood also has *some* capabilities of using **raw HTTP** requests.

1.3 Dependencies

Flood uses several awesome libraries that are listed below. Since Flood is currently in development, *no particular stable versions are required* and by default the newest available versions are pulled in.

- Ibrowse - an HTTP client, found here.
- Lager - a logging framework, found here.
- Folsom - a metrics system, found here.
- JSONx - a fast JSON parser, found here.
- Jesse - a JSON Schema validator, found here.
- `websocket_client` - a WebSocket client, found here.

2 Inner workings

This section describes what happens behind the scenes in **Flood** and how it reflects its usage.

2.1 Simulated Users

- FSMs
- State transitions

2.2 User sessions

2.2.1 Session selection

- Roulette algorithm

2.2.2 Session inheritance

- Single inheritance ordering.
- Multiple inheritance ordering.
- Why so OOP?

2.2.3 Actions & Event handlers

- $\text{on}_{\text{socketio}}$
- on_{event}
- $\text{on}_{\text{timeout}}$

2.3 Flood phases

- Phases purpose
- Phase goals

3 Flood scenarios

This section describes the Flood scenario files and gives some general guildelines for writting them. Example scenarios can be found here.

3.1 Scenario file

Flood uses JSON to encode test scenarios, no Erlang is required. Each scenario resides in a separate file and optionally several goal files (described in detail later). The overall structure of a Flood scenario consists of three required sections:

```
{
    "server" : {
        // Server setup.
    },

    "phases" : {
        // Test phases & goals.

        "phase_I" : {
            ...
        },
        ...
    },

    "sessions" : {
        // User session descriptions.

        "session_A" : {
            ...
        },
        ...
    }
}
```

3.2 Server setup

The `server` section is rather straightforward; it is used to setup the server connection. It has to define several mandatory fields:

```
"server" : {
    "host" : "",     // The server host.
    "port" : 0,      // The server post.
    "endpoint" : "", // Endpoint used to connect to.
    "metadata" : {}  // Server-wide metadata (optional).
}
```

Example server configuration that will cause Flood to connect to http://localhost:80/socket.io/1/ and define some server-wide metadata (more on metadata can be found here):

```
"server" : {
    "host" : "localhost",
    "port" : 80,
    "endpoint" : "/socket.io/1/",
    "metadata" : {
        "foo" : "bar"
    }
}
```

3.3 Phases setup

The `phases` section may define several arbitrarily named Flood phases. The ordering does not matter, as each phase explicitly names its start time.

```
"phases" : {
    "A" : {
        // A's description.
    },

    "B" : {
        // B's description.
    },
    ...
}
```

Each phase description has to follow this format:

```
"phase_I" : {
    "users" : 0,          // Number of users spawned during this phase.
    "user_sessions" : [], // Sessions spawned users should follow.

    "start_time" : 0,     // Time (in milliseconds) at which to start this phase.
    "spawn_duration" : 0, // Duration (in milliseconds) Flood should take to spawn the users.

    "goal" : {},          // Goal of this phase (optional).
    "test_interval" : 0,  // Interval (in milliseconds) of the goal checks (optional).
    "timeout" : 0,        // Timeout (in milliseconds) of this phase (optional).

    "metadata" : {}       // Phase-wide metadata (optional).
}
```

The meaning of each of the fields is as follows:

- `users` - an integer number of users spawned during this phase. It is **mandatory**.
- `user_sessions` - a array of Flood user session names; the concrete user session will be selected at **random according to a sessions weight** (more about this can be found here). It is **mandatory**.
- `start_time` - an integer value that names a point in time (**in milliseconds**), relative to the start of the Flood, at which a phase should be started. It is **mandatory**.
- `spawn_duration` - an integer value that tells Flood how much time (**in milliseconds**) it should take to spawn `users` number of users. Users are spawned uniformly throughout this duration. Keep in mind that for various performance related reasons Flood **may actually take longer** to spawn the users, however it will never take less time to do so. This field is **mandatory**.
- `goal` - either an arbitrary JSON term that is a description of the goal of this phase (more on goals can be found here) or a string containing a path to the file containing the goal description relative to scenario file. This field is **optional**; not defining it will result in no goal checking whatsoever.
- `test_interval` - an integer value that tells Flood at what intervals (**in milliseconds**) in should check whether the `goal` has been reached. It is **optional**; not defining it will result in a single check at the phase `timeout`.
- `timeout` - an integer value that names a point in time (**in milliseconds**), relative to the start of the Flood, at which a phase should be terminated if it is still running. It is **optional**.
- `metadata` - a JSON object defining some phase-wide metadata (more on metadata later). It is **optional**.

Example `phases` setup:

```
"phases" : {
    "phase_I" : {
        "metadata" : { },

        "users" : 1000,
        "user_sessions" : ["session_A", "session_B"],

        "start_time" : 1000,
```

```
        "spawn_duration" : 1000
    },

    "phase_II" : {
        "metadata" : { },

        "users" : 1000,
        "user_sessions" : ["session_C"],

        "start_time" : 2000,
        "spawn_duration" : 5000

        "goal" : "./goal.jsonschema",
        "test_interval" : 100,
        "timeout" : 10000
    }
}
```

This setup will schedule two Flood phases. The first phase, `phase_I`, will start at 1000 ms and spawn 1000 users following either `session_A` or `session_B` over 1000 ms duration. The second phase, `phase_II`, will start at 2000 ms and spawn 1000 users following `session_C` over 5000 ms duration. Additionally, a `phase_II` goal check will be scheduled every 100 ms starting at 2000 ms and running util the goal provided in "./goal.jsonschema" file is met or until the phase timeout, set at 10000 ms, is reached.

3.4 User session setup

- Weights & transports
- Session inheritance
- Actions

3.4.1 Available actions

- Action - arguments - efects - examples list

3.4.2 Timers & Counters

- Starting/stopping/restarting timers
- Managing counters

3.5 Metadata

- Metadata ordering
- Introducing new metadata
- JSON $ubstitutions

3.6 Example scenarios

- Sessions
- Single ping
- Continuous ping
- Simulated “3rd party” requests

4 Flood results

4.1 Results format

- JSON structure
- Counters
- Timers
- Available statistics

4.2 Goal schemas

- JSON Schema structure
- Testing intervals
- Reaching goals
- Goal timeouts

4.3 Continuous Integration integration

- Running Flood automagically