Constellation

Constellation is the new daemon whose purpose is to unite all stateful function of the website, such as liveloading. This involves:

- Deprecating the loved/hated Sails socket on **realtime.beam.pro** and using good ol' HTTP (well, HTTP/2) for API calls. This drastically simplifies many parts of our stack and debugging.
- Creating a new websocket service on **constellation.beam.pro** which handles liveloading and other statefulness.

The current liveloading socket on **realtime.beam.pro** will remain in place until **October 1st**, at which point it will be removed from Beam.

Session

Users authenticate to Constellation by sending their existing cookies in their socket handshake. Constellation will unseal them, like Hapi does, and automatically create a context for that user. No external API endpoints needed.

Users remain connected to Constellation throughout their session on the site. During the session, they may subscribe to events happening on the site (liveloading), get notifications when those events occur.

Constellation Protocol

The protocol is based on JSON-RPC with additional support for an event system. Initially a user handshakes with **constellation.beam.pro**. Aside from the usual handshake headers, following headers may be passed:

- **cookie:** may contain the Beam session cookie. Its presence will cause Constellation to attempt to authenticate the user.
- **authorization:** may contain and OAuth Bearer token to authenticate with for 3rd party apps, rather than using a cookie. This will indicate to Constellation that the user is a bot.
- **x-sec-websocket-protocol**: if set to **cnstl-gzip**, Constellation may choose to send websocket frames down as binary, gzipped JSON rather than plain text. The client may detect gzipped frames by the fact that they are binary messages and begin with the magic bytes **0x1f** and **0x8b** as the first and second payload byte, respectively.
- **x-is-bot**: this **must** be set to true if the client is an automated bot rather than a human user and you are not using an **authorization** header. Failure to set this may cause the account to be banned.

There are three packet types: **request**, **reply**, and **event**. These are sent over the websocket as JSON messages. Messages to the client may be gzipped if **x-supports-gzip** was passed in the handshake, and messages sent to Constellation are always allowed to be gzipped.

- **method** packets are sent from the client in a way very similar to JSON-RPC. This is the only packet the client may send to the server. A packet may look like the following:

  ```
  {"type": "method", "method": "divide", "params": {"numerator": 16, "denominator": 4}, "id": 123}
  ```

  - **type** is always set to "method"
  - **method** should be the name of the method to call
  - **params** should be an *object*, not an array, of named arguments to pass into the method.
  - **id** may be any uint32. It's included in the reply packet and used to correlate replies from the socket.

- **reply** packets are sent in response to method packets. Replies are always sent in response to methods unless the socket closes before they may be sent. A some reply packets may look like the following:

  ```
  {"type": "reply", "result": 4, "error": null, "id": 123}
  {"type": "reply", "result": null, "error": {"code": 1000, "message": "Cannot divide by zero"}, "id": 124}
  ```

    - **type** is always set to "reply"
    - **id** is set to the id passed in the corresponding **method** packet
    - **result** is the unstructured result of the method, or **null**
    - **error** is an error which occurred, or **null**. If present it will always contain a "code" and an associated "message"

      Note that if fatal errors occur as a result of a method call, a websocket close frame will be sent instead of a reply. The close frame's code and associated message will be the same as that which otherwise would have been sent in **reply.error**.

- **event** packets are sent when an action occurs which a client as asked to be notified about. Event packets look like the following:

  ```
  {"type": "event", "event": "math_result", "data": 4}
  ```

    - **type** is always set to "event"
    - **event** is the string name of the event
    - **data** is unstructured information associated with the event

Error Codes

Constellation uses the 4xxx error code range reserved for use by applications in addition to the standard 1xxx codes. The following codes are in use:

- **1011** - Sent in a close or method reply if an unknown internal error occurs.
- **1012** - Sent in a close frame when we deploy or restart Constellation; clients should attempt to reconnect.
- **4000** - Error parsing payload as JSON
- **4001** - Error decompressing a supposedly-gzipped payload
- **4002** - Unknown packet **type**
- **4003** - Unknown method name call
- **4004** - Error parsing method arguments (not the right type or structure)
- **4005** - The user session has expired; if using a cookie, they should log in again, or get a bearer auth token if using an authorization header.
- **4100** - Unknown event in a livesubscribe call
- **4101** - You do not have access to subscribe to the livesubscribe event
- **4102** - You are already subscribed to the livesubscribe event (during **livesubscribe**)
- **4103** - You are not subscribe to the livesubscribe event (during **liveunsubscribe**)

Messages

- A **hello** event is sent down to the client when they first connect.

```
{"type": "event", "event": "hello", "data": {"authenticated": true}}
```

    - **authenticated** is true if the socket consumer authenticated as a user rather than a guest

- A **livesubscribe** method is made available for users to listen to liveloading events.

```
{"type": "method", "method": "livesubscribe", "data": {"events":
["user:1:update", "channel:1:follow"]}, "id": 42}
```

    - **events** is an array of events to subscribe to. Note that either all events are successfully subscribe to, or a failure occurs and no events are subscribed to. Either do or do not, there is no try.

- A **livesubscribe reply** looks like one of the following:
    - A successful reply:
      ```
      {"type": "reply", "result": null, "error": null, "id": 42}
      ```
    - A reply with an invalid event:
      ```
      {"type": "reply", "result": null, "error": {"code": 4100,
      "message": "Unknown event 'my silly event'"}, "id": 42}
      ```
    - A reply for an event you do not have permission to see:
      ```
      {"type": "reply", "result": null, "error": {"code": 4101,
      "message": "Access denied on 'user:1:secrets'"}, "id": 42}
      ```
    - A reply for an event you are already subscribed to:
      ```
      {"type": "reply", "result": null, "error": {"code": 4102,
      "message": "Attempt to duplicate subscription to 'user:1:update"},
      "id": 42}
      ```

- A **liveunsubscribe** method can be used to stop listening to previously-subscribed-to events:

```
{"type": "method", "method": "livesubscribe", "data": {"events":
["user:1:update", "channel:1:follow"]}, "id": 42}
```

    - **events** is an array of events to unsubscribe from. Note that if you are not subscribed to one or more of the events, no error is returned.
- A **liveunsubcribe reply** looks like the following:

```
{"type": "reply", "result": null, "error": null, "id": 42}
```

- A **live event** looks like the following. Do note the *socket event names are not liveloading events*. Events you ask for over liveloading are always "live" events which contain the liveloading information. This separation

is added so that other kinds of events can be distinguished from liveloading events.

```
{"type": "event", "event": "live", "data": {"channel": "user:1:update", "payload": {"sparks": 10000}}
```