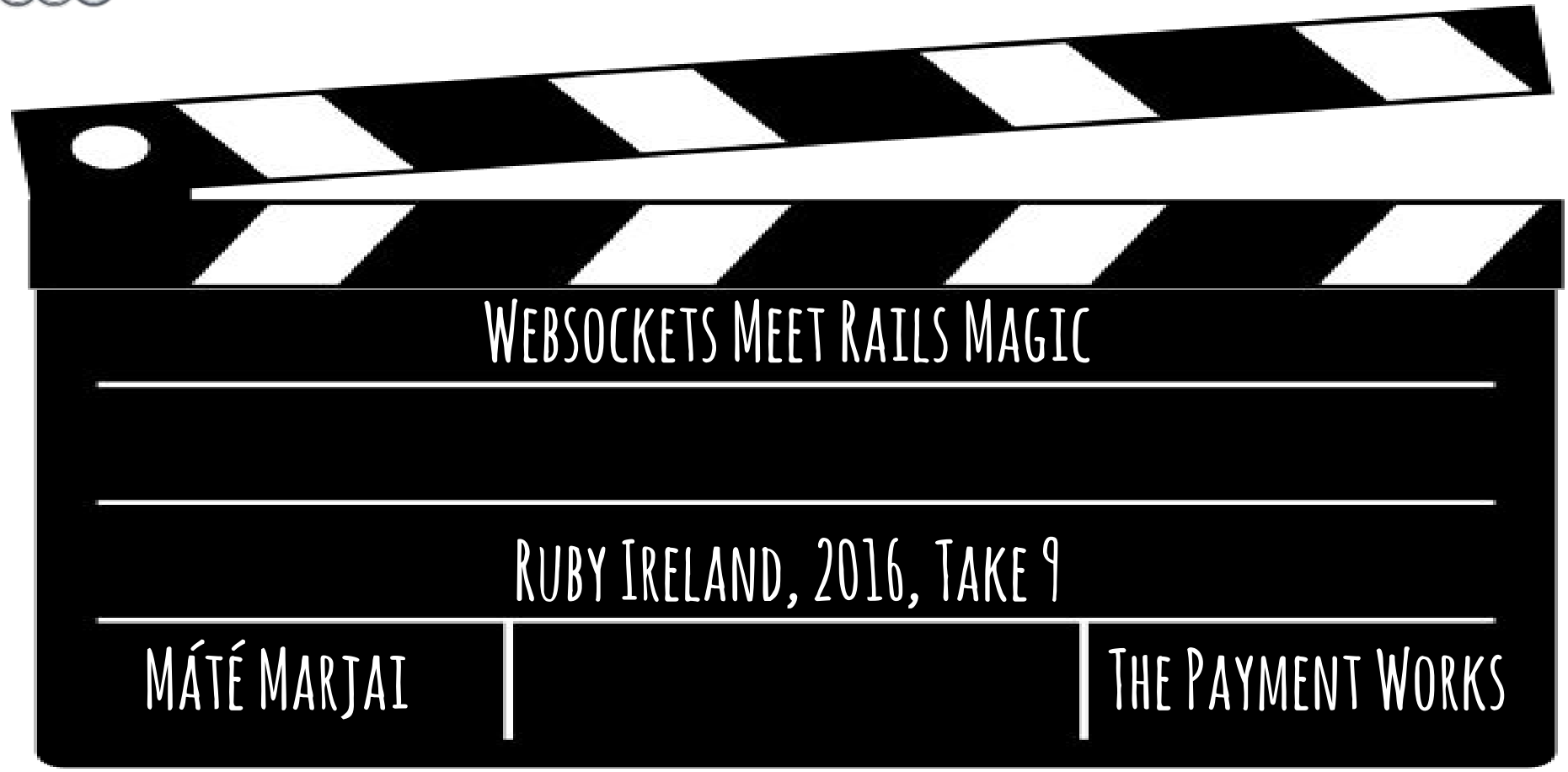

Websockets Meet Rails Magic
Ruby Ireland, 2016, Take 9
Máté Marjai
The Payment Works

Rails 5? Rails 5!

API mode - yes please!

Simpler CLI commands - bring it on!

SQL ‘or’ with ActiveRecord - finally!

ActionCable - I don’t know what that is, but has ‘Action’ in the name, so must be good!

Action Cable - not for your netflix

Fully integrated websockets into your Rails app

Full access to your session and domain models - as you'd expect with a Rails app.

Comes with a client-side wrapper to make it easier to implement and handle.

Other Websocket gems

- Websocket-ruby
- Event Machine Websockets
- Skinny - for thin-based solutions
- Websocket-driver

So, what are websockets?

Websockets!

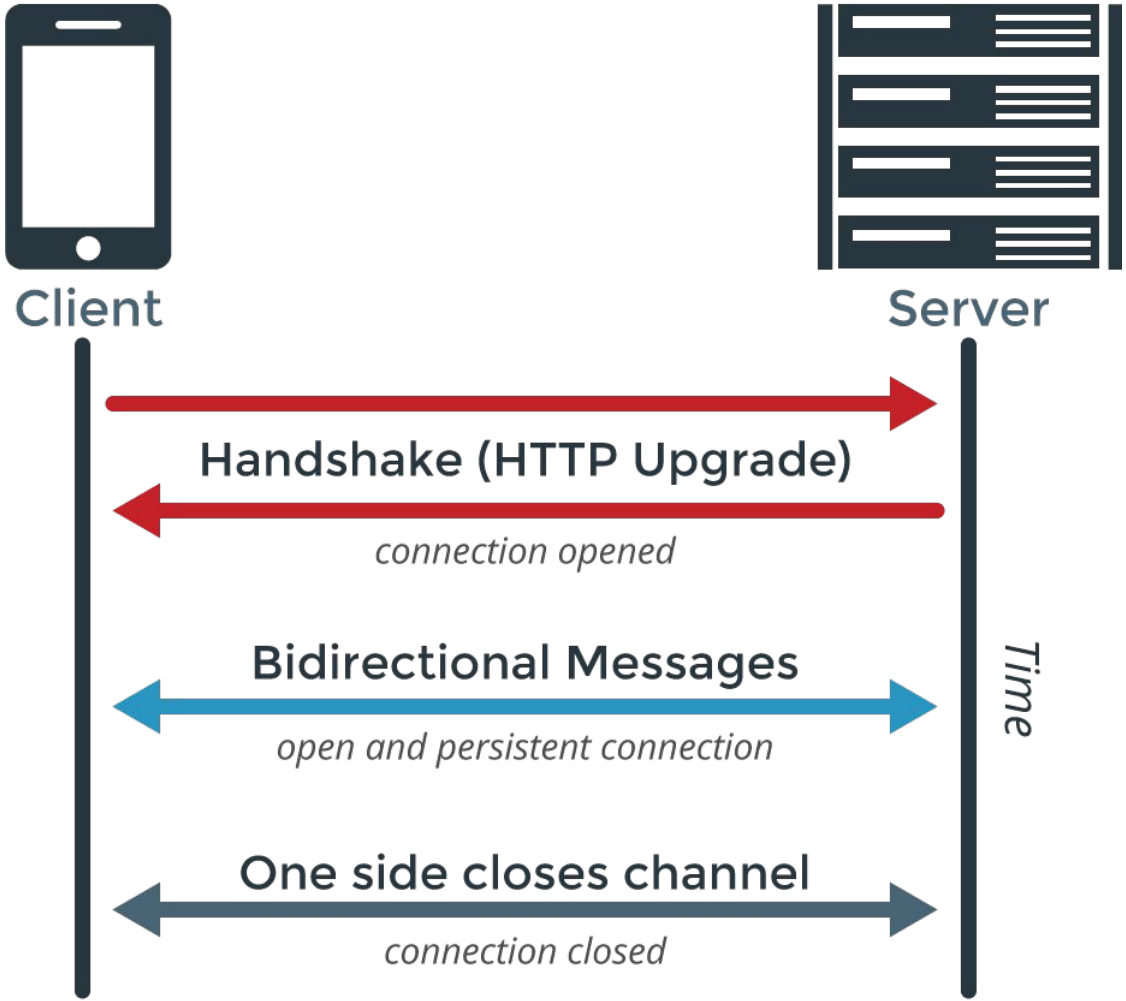
WebSocket is a protocol providing full-duplex communication channels over a single TCP connection.

It’s a client - server solution to send and receive messages on designated channels.

Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.

HTTP Upgrade

```
Incoming Headers:

  Origin: http://localhost:4200

Started GET "/cable" for 127.0.0.1 at 2016-09-14 20:26:21
+0100

Started GET "/cable/" [WebSocket] for 127.0.0.1 at
2016-09-14 20:26:21 +0100

Successfully upgraded to WebSocket (REQUEST_METHOD: GET,
HTTP_CONNECTION: Upgrade, HTTP_UPGRADE: websocket)

Channel is transmitting the subscription confirmation
```


Client
Server
Handshake (HTTP Upgrade)
connection opened
Bidirectional Messages
open and persistent connection
One side closes channel
connection closed
Time

Send me stuff, but don't make me ask for it!

Websockets are great for Pub/Sub services:

- One consumer, many subscriptions
- Each subscription has its own channel and message queue
- You can send streams of data
- Or occasional messages

Gotchas

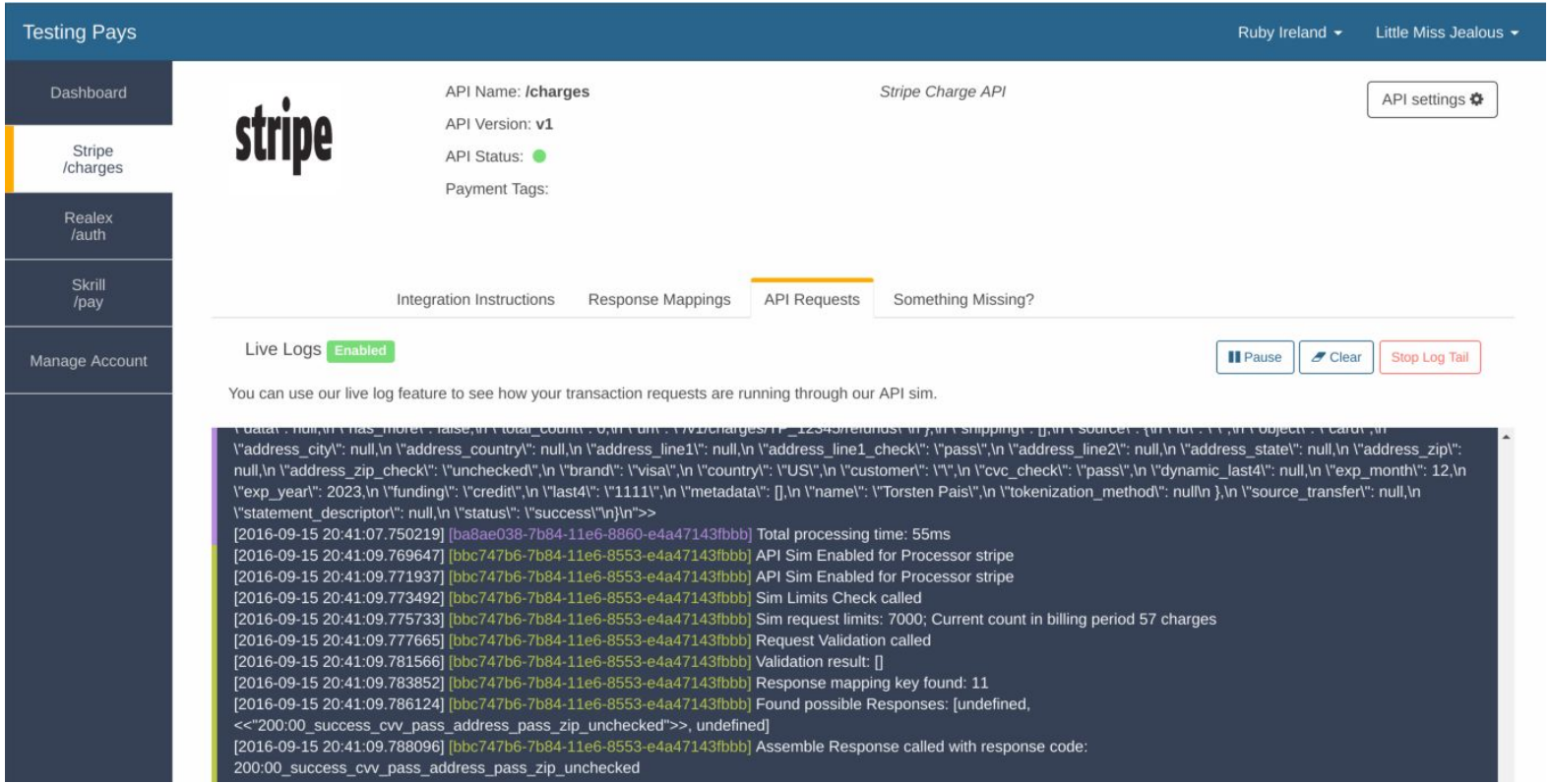
Make sure you are using **puma** or **unicorn** (or similar) when running Rails with ActionCable or an ActionCable server alone.

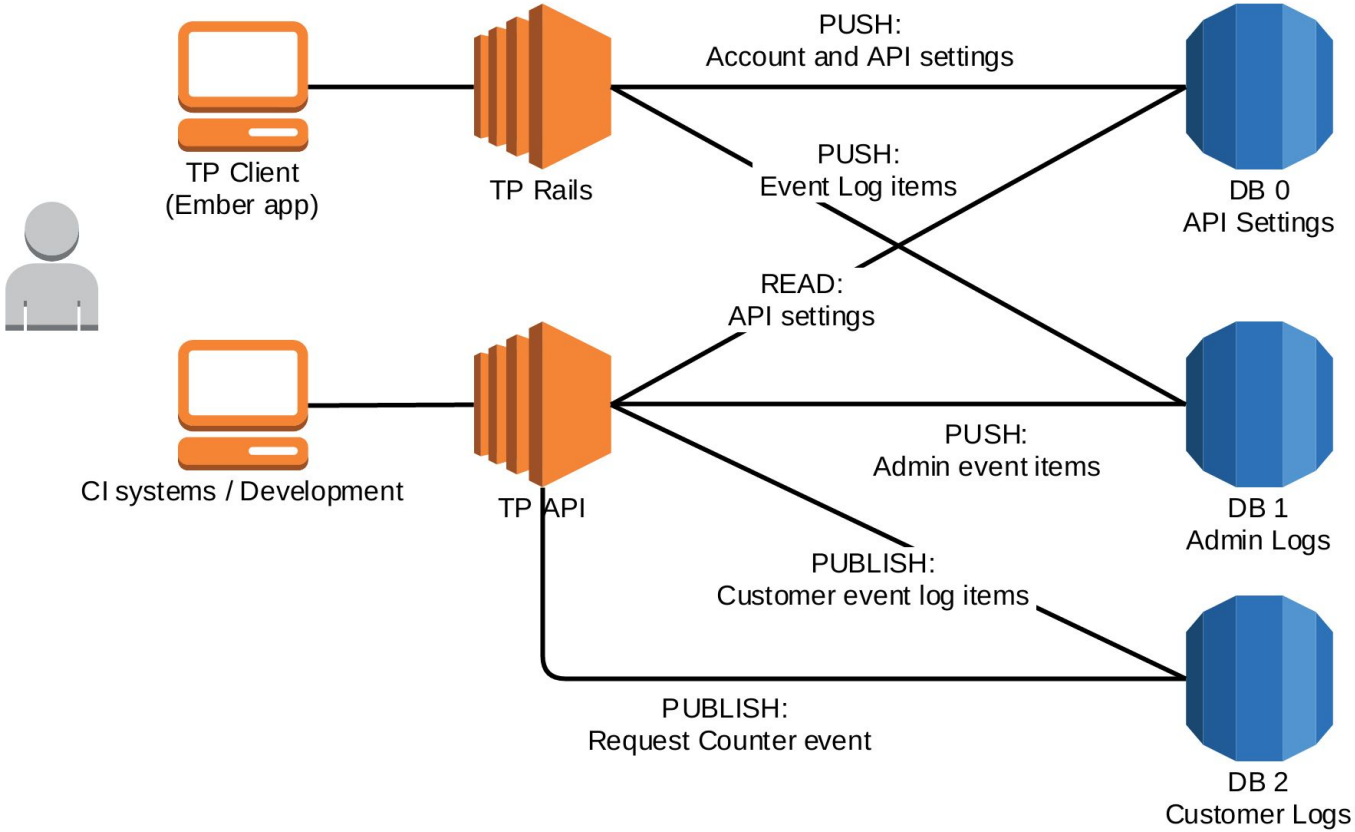
Don't forget the upgrade part of HTTP, that still needs to be added to your server config.

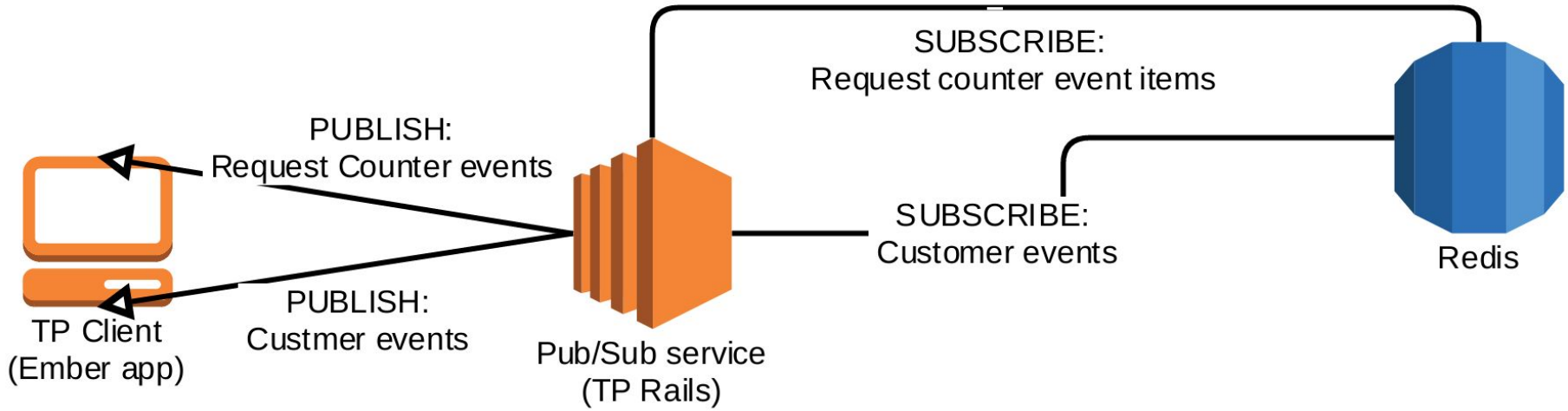
Non-Ruby alternatives

- Socket.io - Node
- Elixir-Socket - Elixir
- Phoenix - Elixir
- Ws - Node

Testing Pays




TP Client
(Ember app)
TP Rails
PUSH:
Account and API settings
PUSH:
Event Log items
DB 0
API Settings
READ:
API settings
CI systems / Development
TP API
PUSH:
Admin event items
DB 1
Admin Logs
PUBLISH:
Customer event log items
PUBLISH:
Request Counter event
DB 2
Customer Logs


SUBSCRIBE:
Request counter event items
PUBLISH:
Request Counter events
SUBSCRIBE:
Customer events
Redis
PUBLISH:
Custmer events
TP Client
(Ember app)
Pub/Sub service
(TP Rails)

Server code

```
class LogTailChannel < ApplicationCable::Channel
 # Subscribe to Redis channel
 def subscribed
   if params[:api_key] && params[:processor_name] && params[:api_name]
     stream_from "live_log_#{params[:processor_name]}_#{params[:api_name]}_#{params[:api_key]}"
   end
 end
 # Stop streams
 def unsubscribed
   stop_all_streams
 end
end
```

Client code - setup the consumer using ember-cable

```
const ApiLiveLogComponent = Ember.Component.extend({
 // --------- snip ----------------
 setupConsumer: Ember.on('didInsertElement', function() {
   // Only setup the channel consumer if we have the live log enabled.
   // Using ‘ember-cable’
   if (this.get('currentTeam.live_log_enabled')) {
     this.set('consumer',
this.get('cableService').createConsumer(ENV.streaming_api));
   }
 }),
```

Client code - handling incoming messages

```
// --------- snip ----------------
// Append the log items JSON as they come in through action cable.
logTailDisplay(data) {
  // If the log tail started but paused for some reason, we add items to the buffer
  if (this.get('logTailPaused')) {
    this.get('logItemsBuffer').pushObject(data);
  } else {
    this.get('logItems').pushObject(data);
  }
},
```

Client code - Button control start

```
// --------- snip ----------------
actions: {
  // Start tailing, i.e. subscribe to the channel and start displaying the
  // logs as they come in through the websocket.
  startTail: function() {
    this.set('subscription', this.get('consumer').subscriptions.create({
        channel: 'LogTailChannel',
        api_key: this.get('currentTeam.api_key'),
        processor_name: this.get('model.api.payment_processor.keyword'),
        api_name: this.get('model.api.name')
      }, {
      received: (data) => { this.logTailDisplay(data); }
    }));
  },
```

Client code - Button control stop

```
// --------- snip ----------------
actions: {
  // Stop the log tail. i.e. unsubscribe and disconnect from the websocket.
  stopTail: function() {
    this.get('subscription').unsubscribe();
    this.set('subscription', null);
    // Reset the log tail paused state.
    this.set('logTailPaused', false);
  },
```

Client code - Button control Pause / Play

```
// --------- snip ----------------
actions: {
  // Allow users to pause & play the logs as they come in.
  toggleLogTail: function() {
    this.toggleProperty('logTailPaused');
    // IF tail is in play > add the buffer to the mail log items & clear.
    this.get('logItems').pushObjects(this.get('logItemsBuffer'));
    this.get('logItemsBuffer').clear();
  }
}
```

Live demo