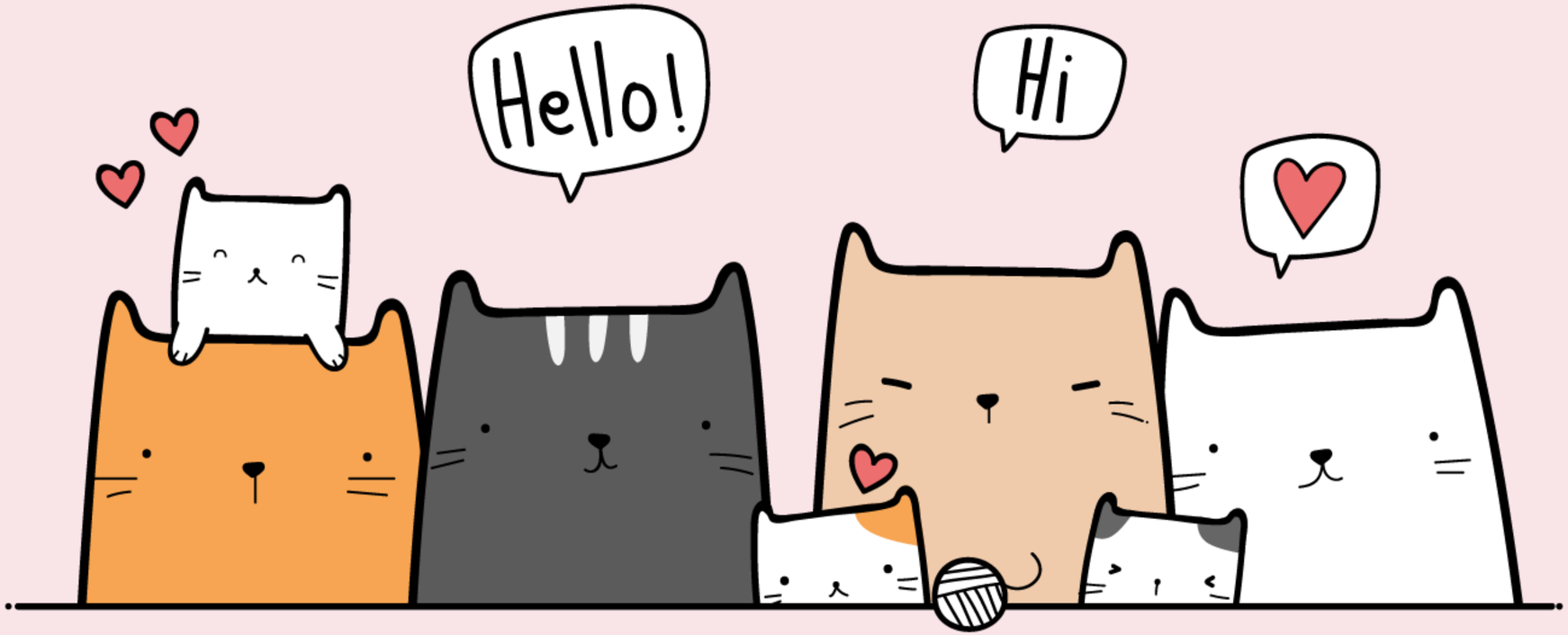
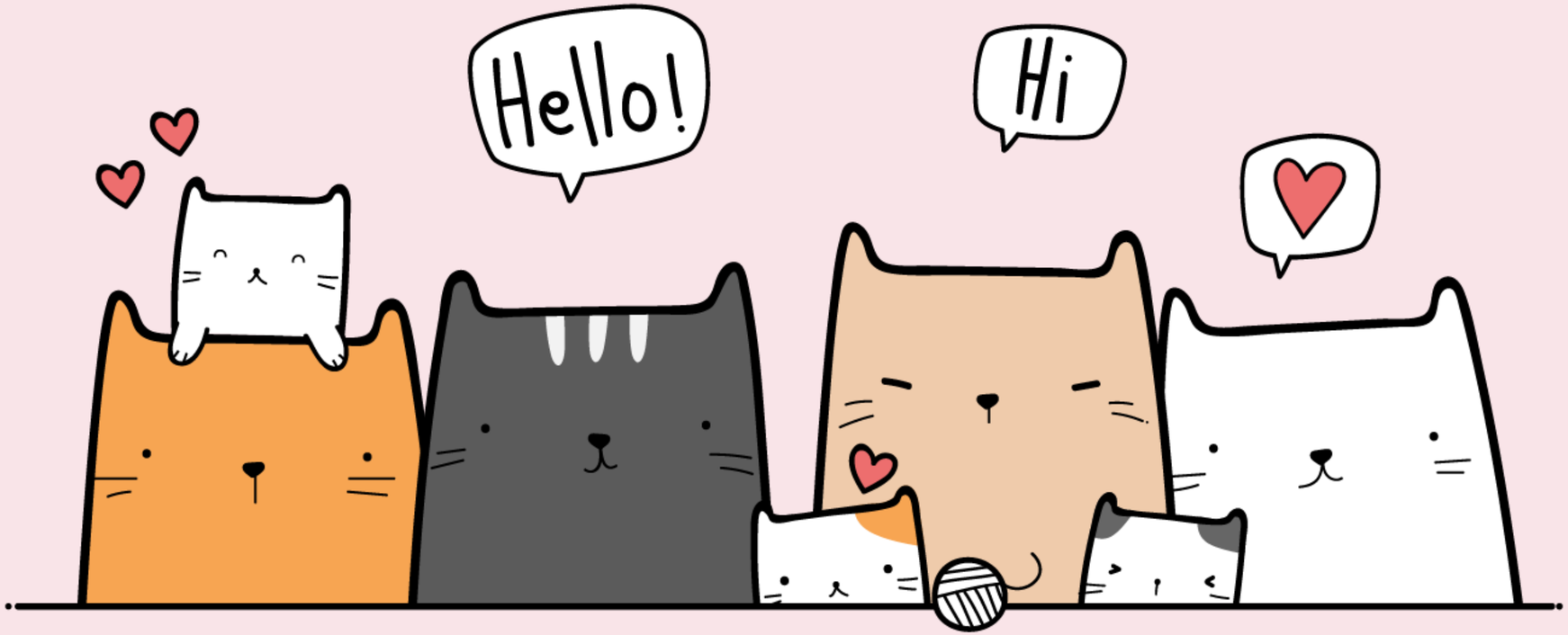


Reduce, Reuse, Recycle

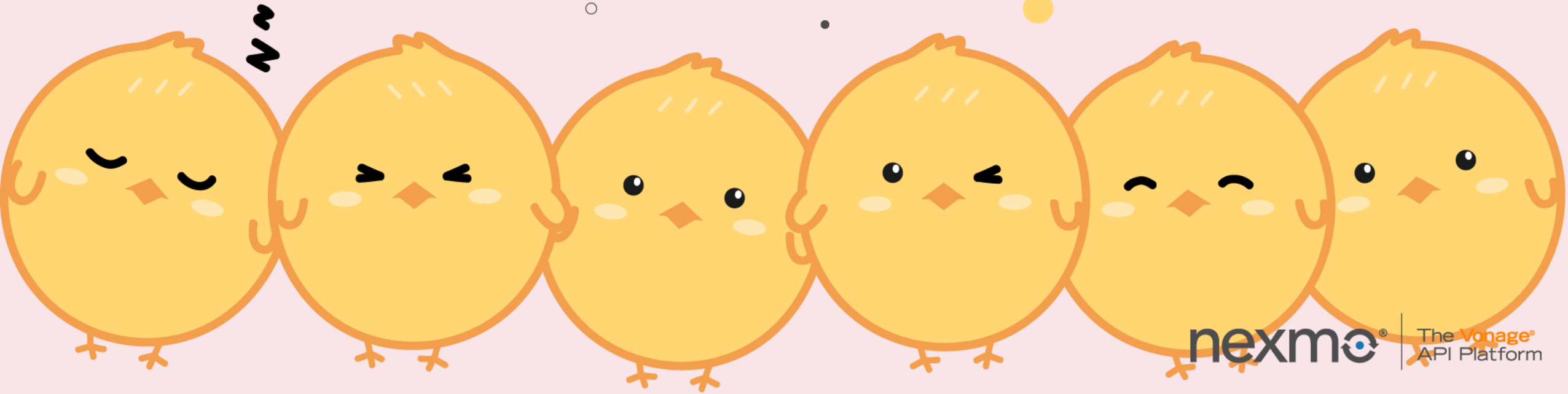
by Aaron Bassett



Reduce, Reuse, Recycle

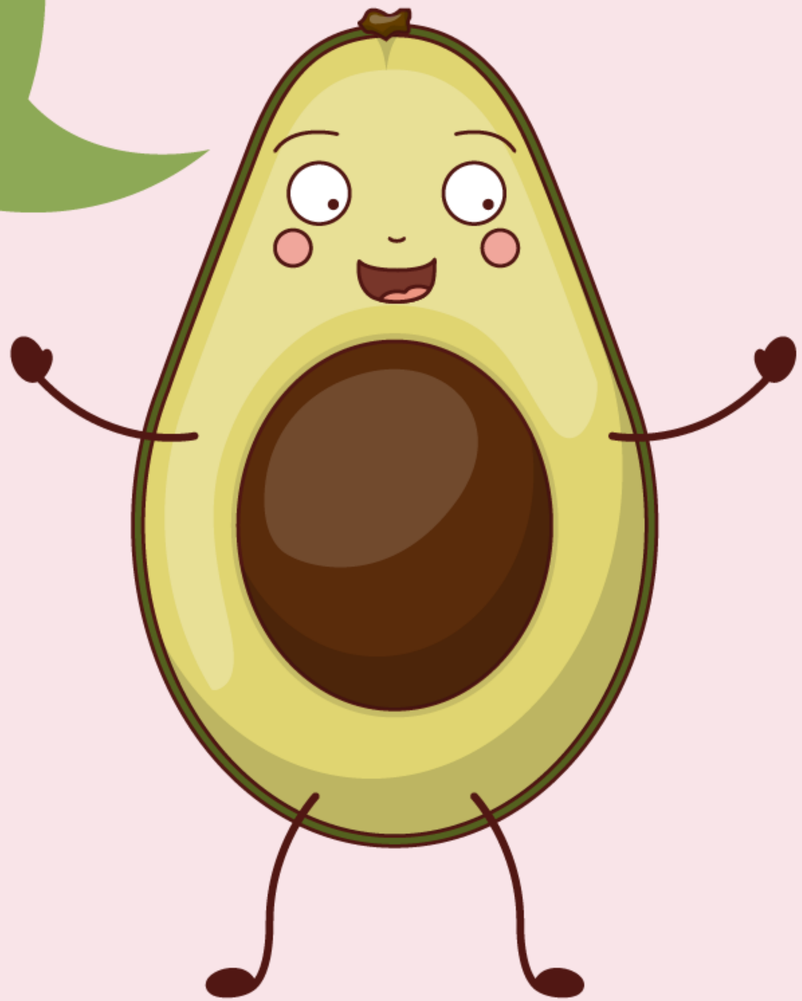
by Aaron Bassett

@aaronbassett



nexmo® | The Vonage® API Platform

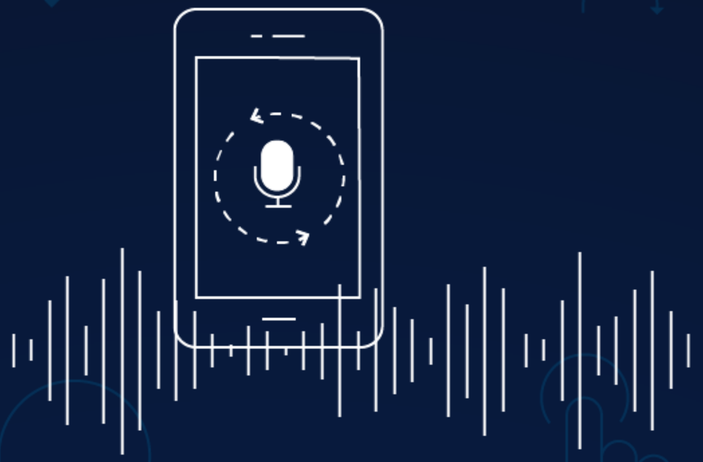
I'm not in Sales or Marketing



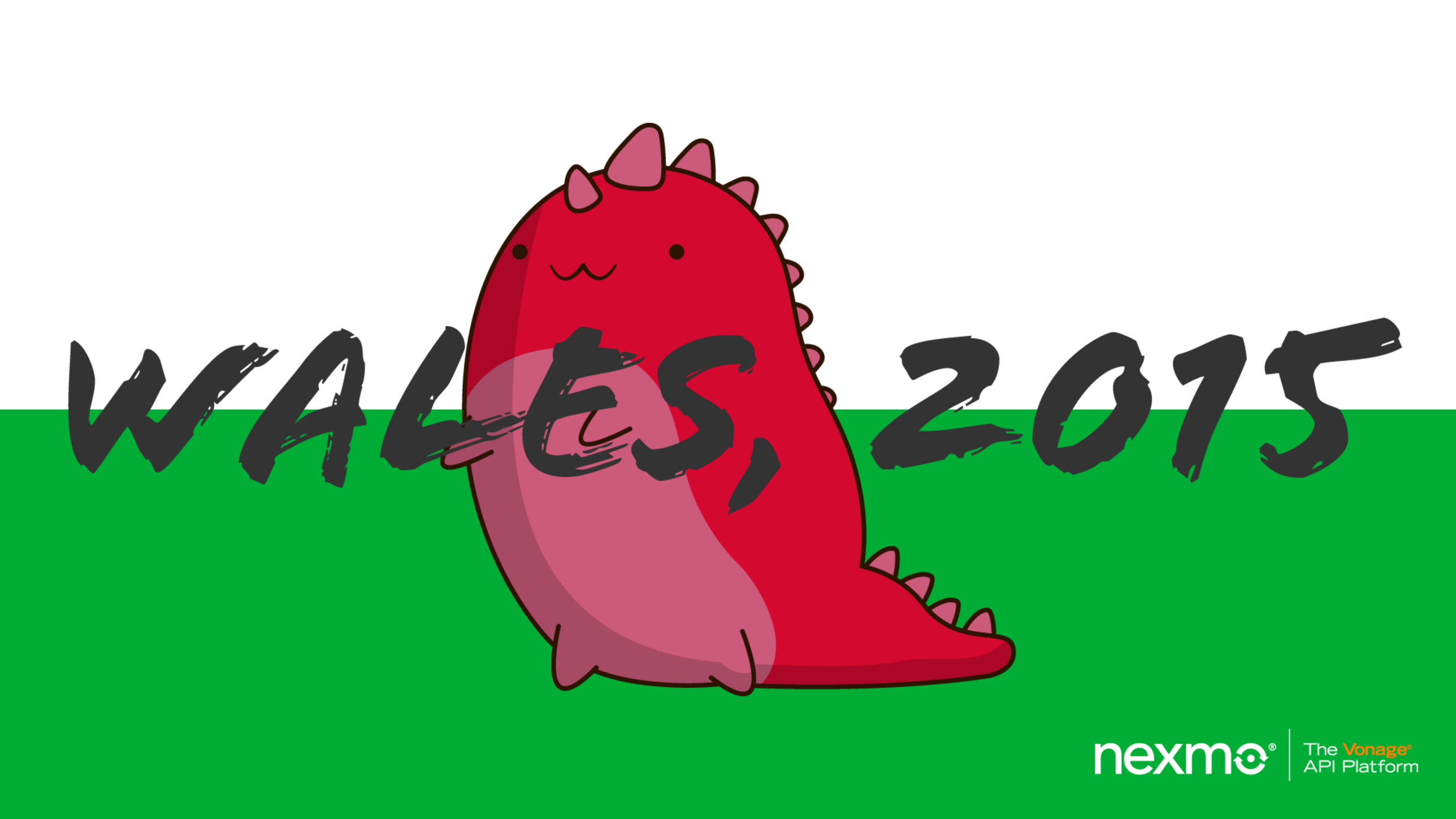
nexmo® | The Vonage® API Platform

nexmo®

The Vonage® API Platform



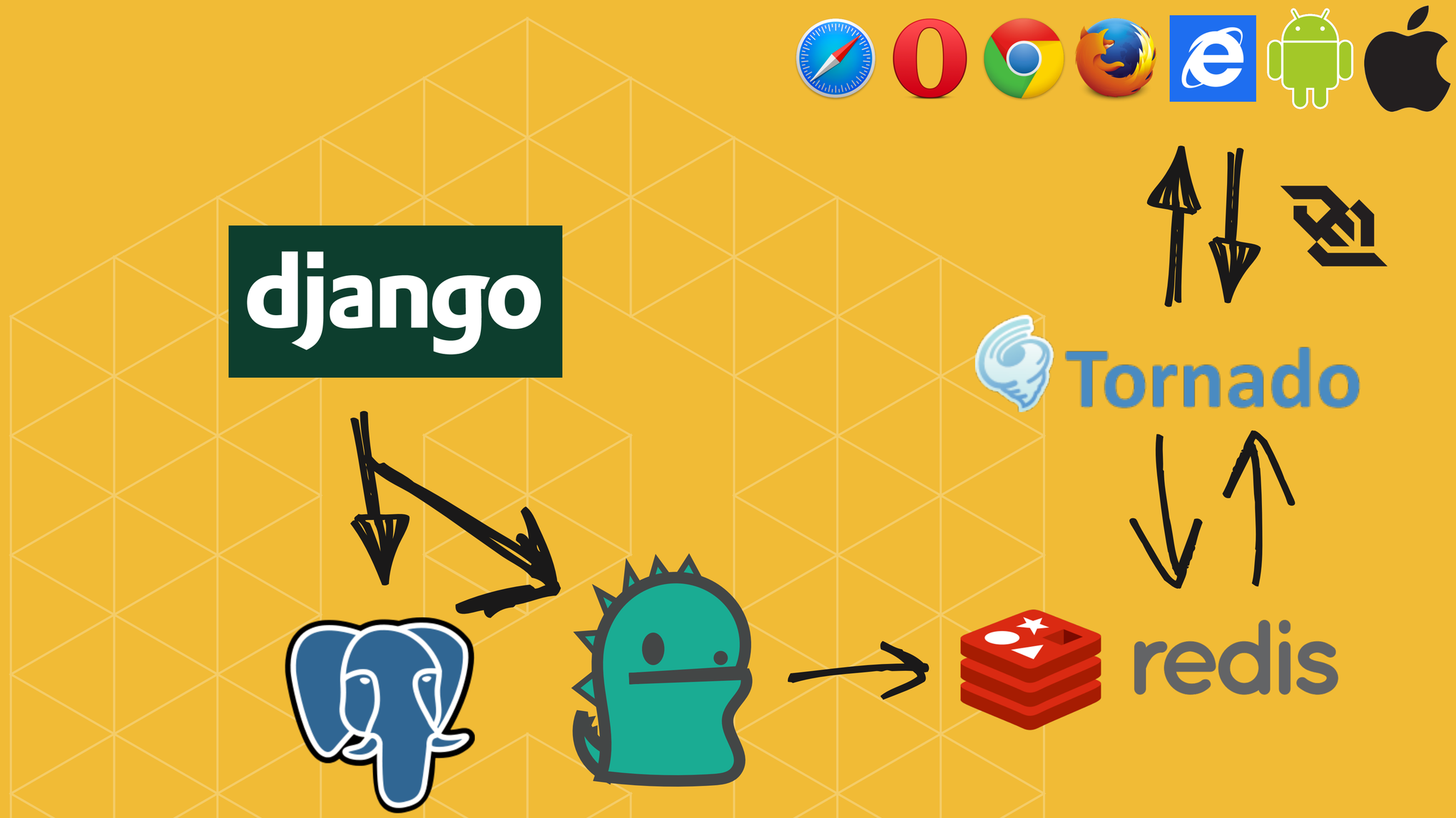
nexmo® | The Vonage® API Platform

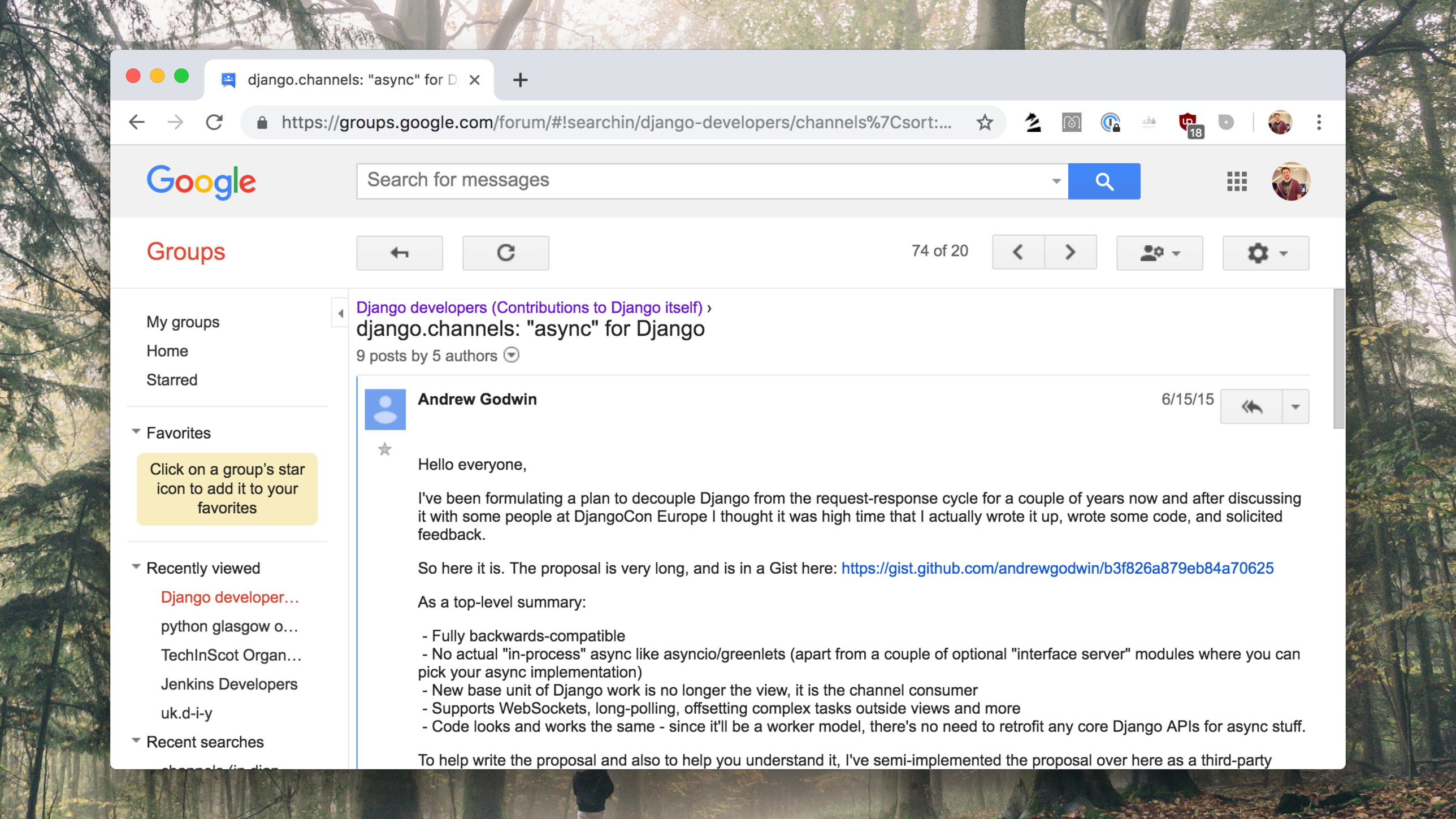


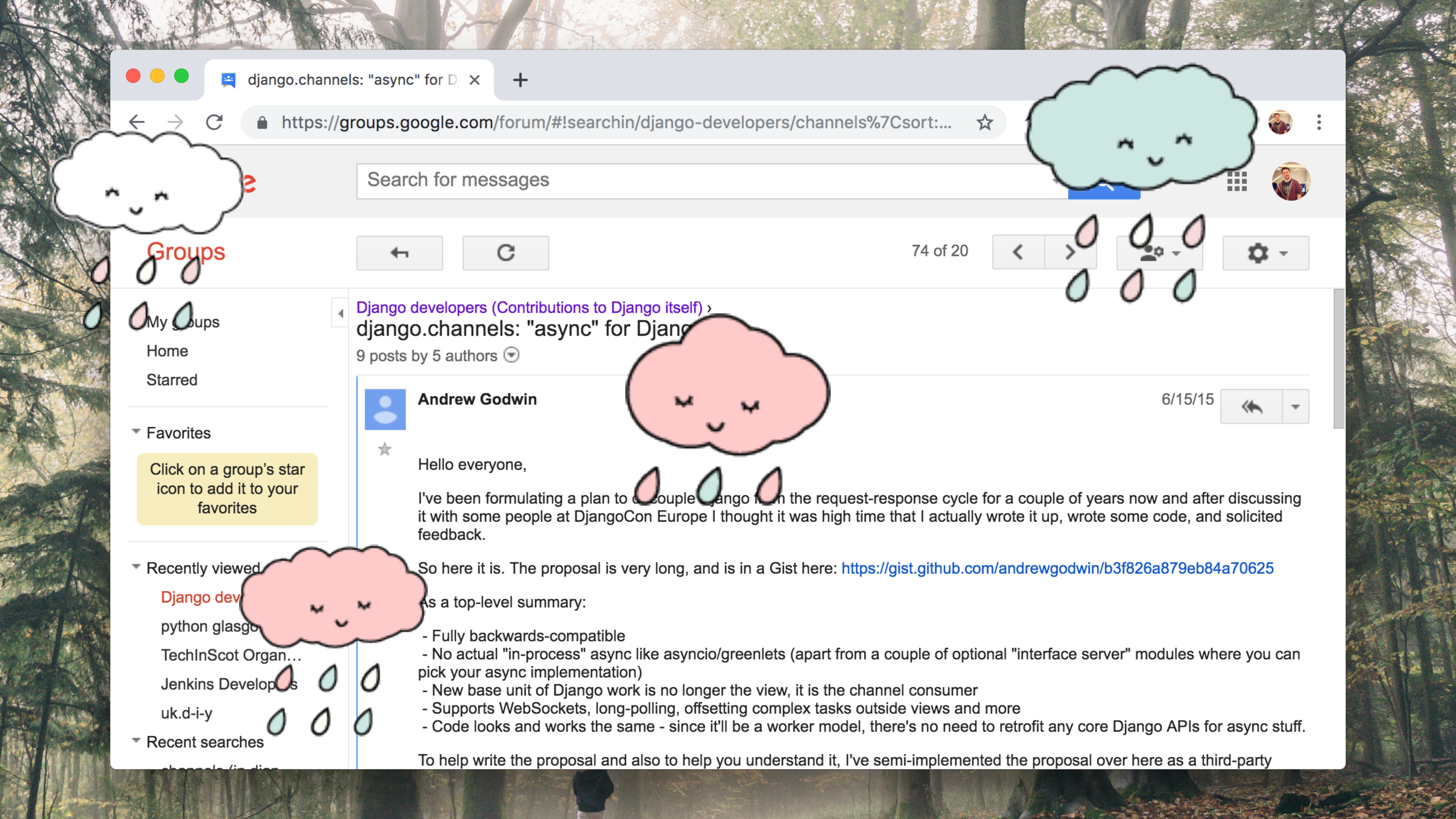
nexmo® | The Vonage® API Platform

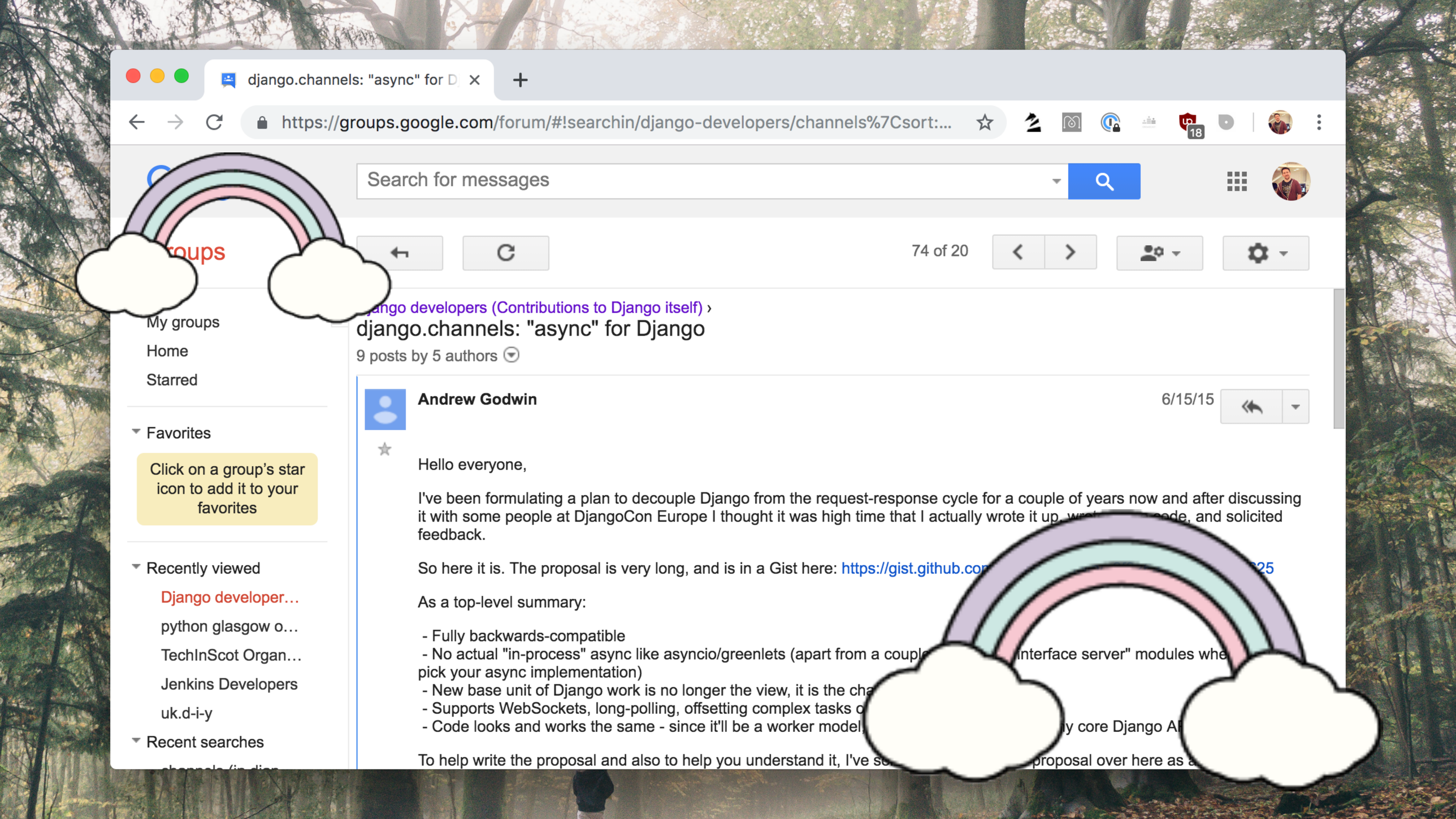
Effortless real time apps in Django

@aaronbassett – rawtech.io

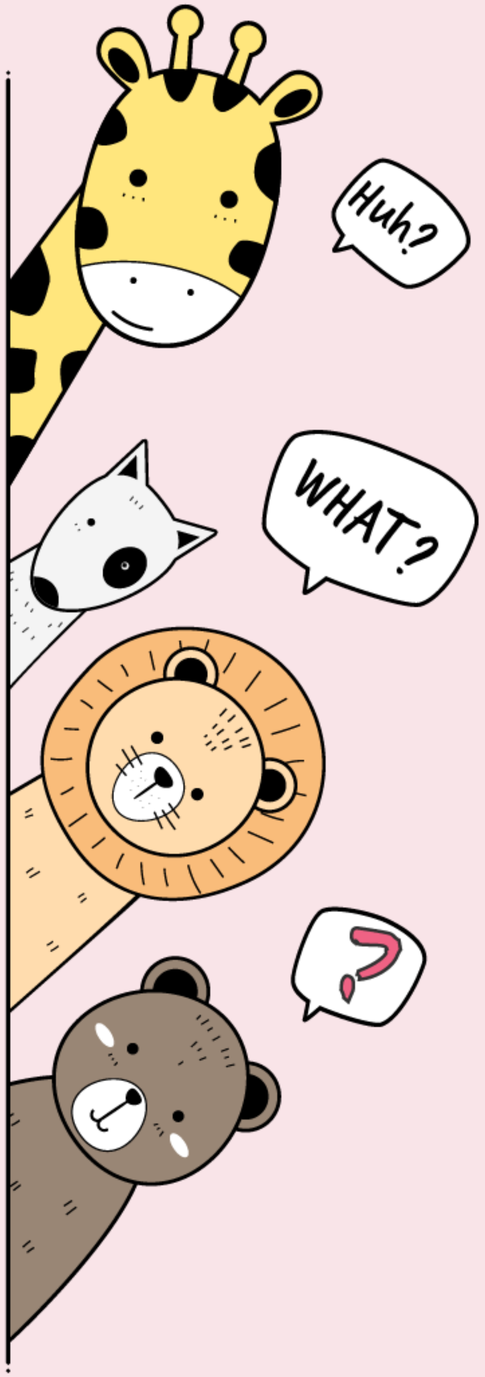

django
Tornado
redis


django.channels: "async" for D
https://groups.google.com/forum/#!searchin/django-developers/channels%7Csort:...
Google
Search for messages
Groups
74 of 20
My groups
Home
Starred
Favorites
Click on a group's star icon to add it to your favorites
Recently viewed
Django developer…
python glasgow o…
TechInScot Organ…
Jenkins Developers
uk.d-i-y
Recent searches
Django developers (Contributions to Django itself) ›
django.channels: "async" for Django
9 posts by 5 authors
Andrew Godwin
6/15/15
Hello everyone,
I've been formulating a plan to decouple Django from the request-response cycle for a couple of years now and after discussing it with some people at DjangoCon Europe I thought it was high time that I actually wrote it up, wrote some code, and solicited feedback.
So here it is. The proposal is very long, and is in a Gist here: https://gist.github.com/andrewgodwin/b3f826a879eb84a70625
As a top-level summary:
- Fully backwards-compatible
- No actual "in-process" async like asyncio/greenlets (apart from a couple of optional "interface server" modules where you can pick your async implementation)
- New base unit of Django work is no longer the view, it is the channel consumer
- Supports WebSockets, long-polling, offsetting complex tasks outside views and more
- Code looks and works the same - since it'll be a worker model, there's no need to retrofit any core Django APIs for async stuff.
To help write the proposal and also to help you understand it, I've semi-implemented the proposal over here as a third-party

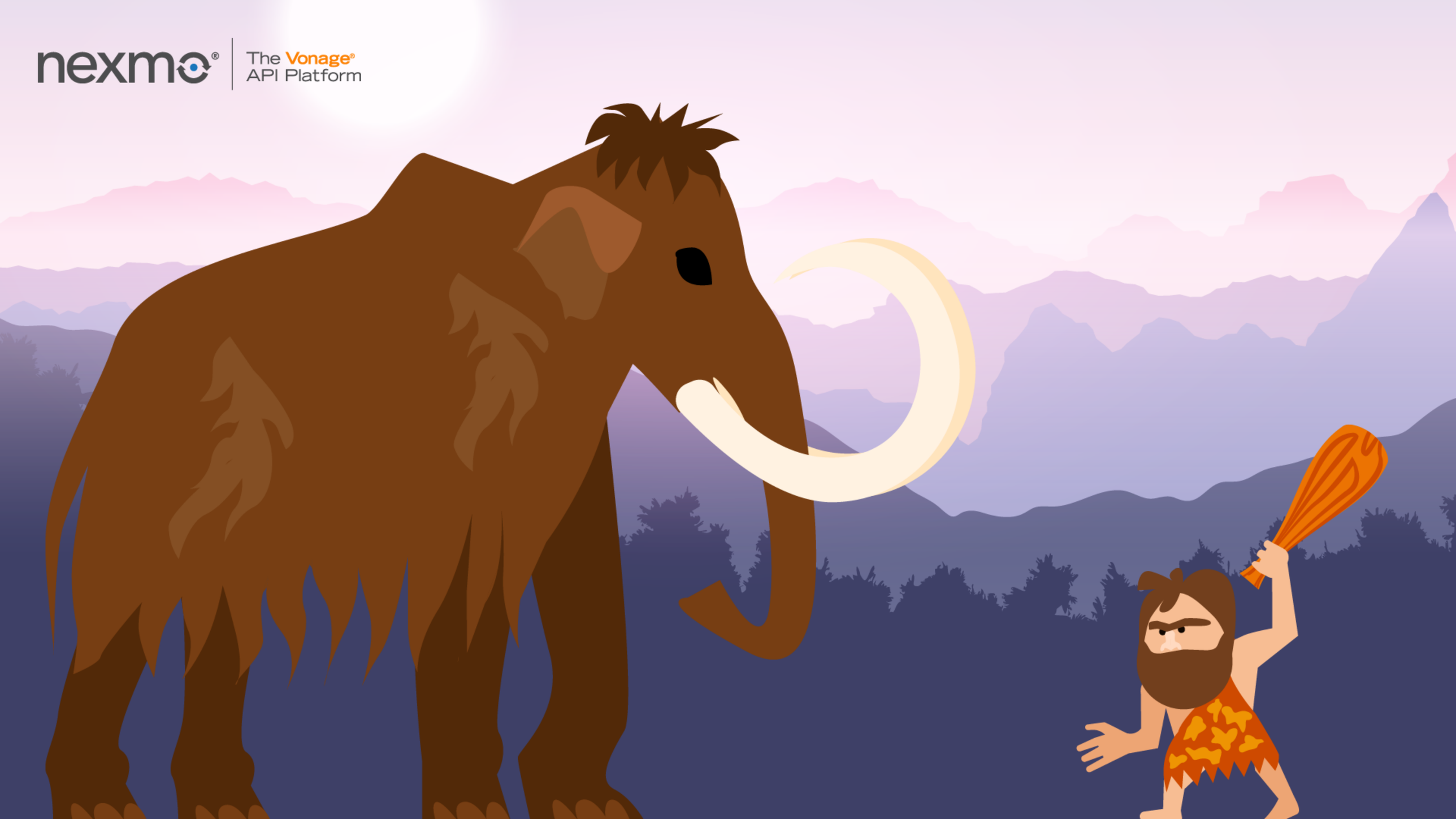

django.channels: "async" for D
https://groups.google.com/forum/#!searchin/django-developers/channels%7Csort:...
Search for messages
Groups
74 of 20
My Groups
Home
Starred
Favorites
Click on a group's star icon to add it to your favorites
Recently viewed
Django dev
python glasgo
TechInScot Organ…
Jenkins Develop s
uk.d-i-y
Recent searches
Django developers (Contributions to Django itself) ›
django.channels: "async" for Djan
9 posts by 5 authors
Andrew Godwin
6/15/15
Hello everyone,
I've been formulating a plan to d couple ango f n the request-response cycle for a couple of years now and after discussing it with some people at DjangoCon Europe I thought it was high time that I actually wrote it up, wrote some code, and solicited feedback.
So here it is. The proposal is very long, and is in a Gist here: https://gist.github.com/andrewgodwin/b3f826a879eb84a70625
As a top-level summary:
- Fully backwards-compatible
- No actual "in-process" async like asyncio/greenlets (apart from a couple of optional "interface server" modules where you can pick your async implementation)
- New base unit of Django work is no longer the view, it is the channel consumer
- Supports WebSockets, long-polling, offsetting complex tasks outside views and more
- Code looks and works the same - since it'll be a worker model, there's no need to retrofit any core Django APIs for async stuff.
To help write the proposal and also to help you understand it, I've semi-implemented the proposal over here as a third-party


django.channels: "async" for D
https://groups.google.com/forum/#!searchin/django-developers/channels%7Csort:...
Search for messages
roups
74 of 20
My groups
Home
Starred
Favorites
Click on a group's star icon to add it to your favorites
Recently viewed
Django developer…
python glasgow o…
TechInScot Organ…
Jenkins Developers
uk.d-i-y
Recent searches
ango developers (Contributions to Django itself) ›
django.channels: "async" for Django
9 posts by 5 authors
Andrew Godwin
6/15/15
Hello everyone,
I've been formulating a plan to decouple Django from the request-response cycle for a couple of years now and after discussing it with some people at DjangoCon Europe I thought it was high time that I actually wrote it up,
feedback.
So here it is. The proposal is very long, and is in a Gist here: https://gist.github.co
As a top-level summary:
- Fully backwards-compatible
- No actual "in-process" async like asyncio/greenlets (apart from a coupl
interface server" modules whe
pick your async implementation)
- New base unit of Django work is no longer the view, it is the cha
- Supports WebSockets, long-polling, offsetting complex tasks o
- Code looks and works the same - since it'll be a worker model
y core Django A
To help write the proposal and also to help you understand it, I've s
proposal over here as a

What are WebSockets?



nexmo® | The Vonage® API Platform


nexmo
The Vonage API Platform

nexmo® | The Vonage® API Platform


nexmo
The Vonage API Platform
"My kids are always on AOL Instant Messenger our website should have it's own chat too!"
<iframe>

```
HTTP/1.1 200 OK
  Content-Type: text/html; charset=iso-8859-1
  Transfer-Encoding: chunked
```

```
HTTP/1.1 200 OK
  Content-Type: text/html; charset=iso-8859-1
  Transfer-Encoding: chunked
```

```
33
<script>
  doSomething();
</script>
33
<script>
  doSomething();
</script>
33
<script>
  doSomething();
</script>
```

```
33
<script>
  doSomething();
</script>
33
<script>
  doSomething();
</script>
33
<script>
  doSomething();
</script>
```


Comet
nexmo
The Vonage API Platform

```
(function poll() {
    new Ajax.Request('/api/', {
        method: 'get',
        timeout: 60000,
        onSuccess: function() {
            // Do something
            poll();
        },
        onFailure: function() {
            // Do something else
            poll();
        }
    });
}());
```

```
GET / HTTP/1.1
  Host: 2019.djangocon.eu
  Connection: keep-alive
  Upgrade-insecure-requests: 1
  Dnt: 1
  User-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
  Referer: https://www.google.com/
  Accept-encoding: gzip, deflate, br
  Accept-language: en-GB,en;q=0.9,en-US;q=0.8
```

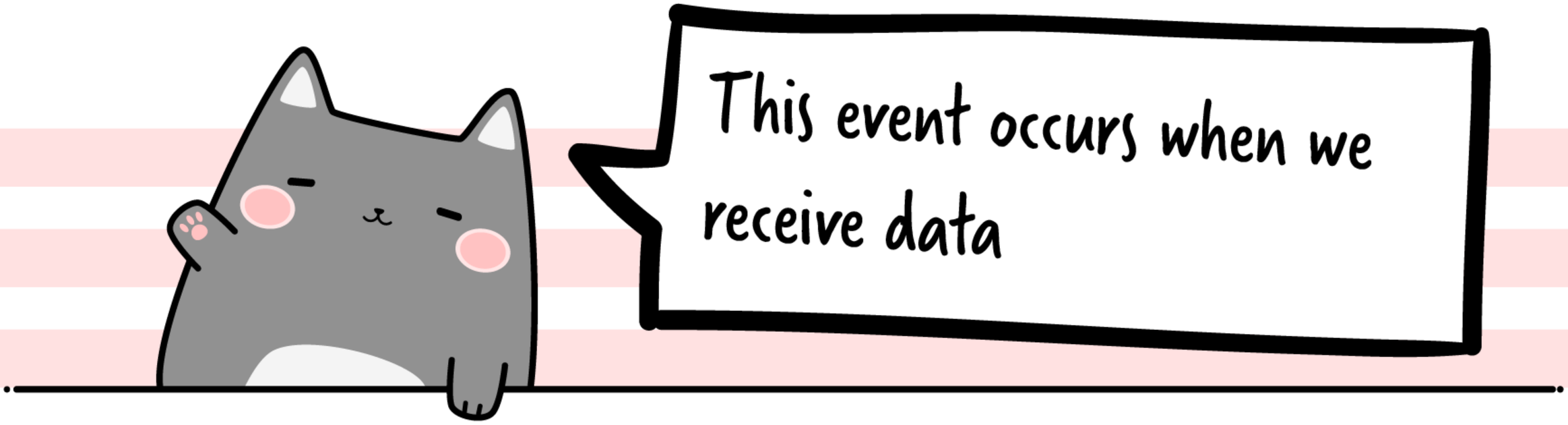
```
HTTP/1.1 200 OK
  Server: nginx/1.14.0 (Ubuntu)
  Date: Wed, 10 Apr 2019 17:18:50 GMT
  Content-Type: text/html
  Last-Modified: Wed, 10 Apr 2019 12:14:03 GMT
  Connection: keep-alive
  ETag: W/"5cadde0b-4c01"
  Strict-Transport-Security: max-age=31536000;
  X-Content-Type-Options: nosniff
  X-Frame-Options: SAMEORIGIN
  Content-Encoding: gzip
```

```
GET / HTTP/1.1
  Upgrade: WebSocket
  Connection: Upgrade
  Host: www.websocket.org
```

```
HTTP/1.1 101 WebSocket Protocol Handshake
  Upgrade: WebSocket
  Connection: Upgrade
```

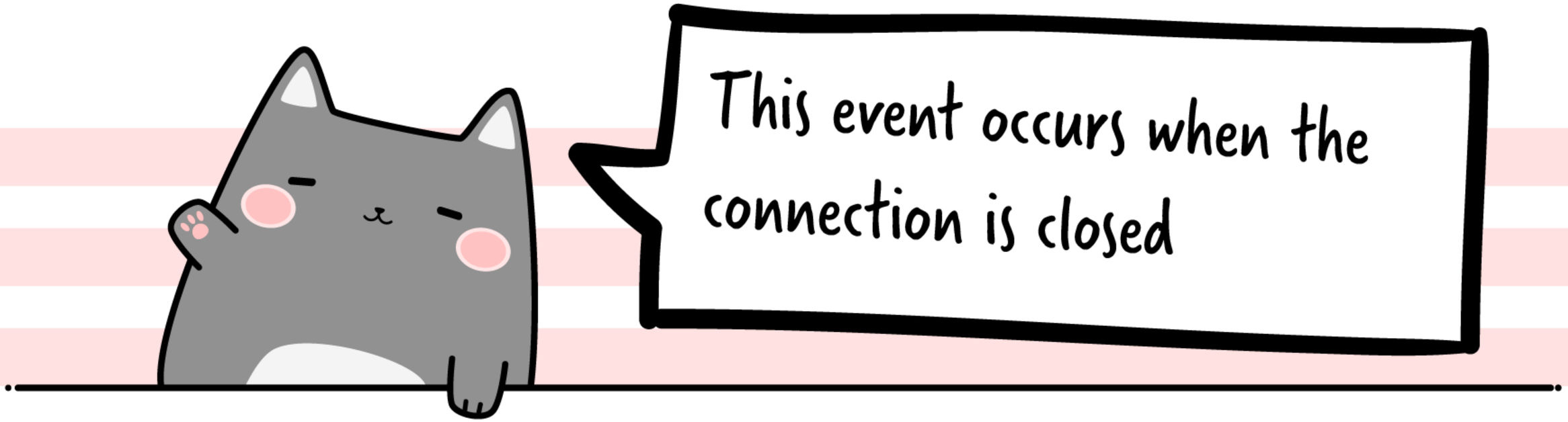

Bidirectional

Socket.onopen



Socket.onmessage

Socket.onerror



Socket.onclose

The send(data) method
transmits data using the connection

Socket.send()

The close() method would be used to terminate any existing connection

Socket.close()

```python
class CountConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = ???
        self.room_group_name = ???
        self.redis_client = redis.Redis(???)

        self.redis_client.incr('connections')
        self.redis_client.set('updated', time.time())

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.channel_layer.group_send(
            self.room_group_name,
            {
                'type': 'connection_message',
                'updated': ???,
                'connections': ???
            }
        )

        await self.accept()
```

```python
class CountConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = ???
        self.room_group_name = ???
        self.redis_client = redis.Redis(???)

        self.redis_client.incr('connections')
        self.redis_client.set('updated', time.time())

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.channel_layer.group_send(
            self.room_group_name,
            {
                'type': 'connection_message',
                'updated': ???,
                'connections': ???
            }
        )

        await self.accept()
```

```python
class CountConsumer(AsyncWebsocketConsumer):
    async def connect(self):
```

```python
class CountConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = ???
        self.room_group_name = ???
        self.redis_client = redis.Redis(???)

        self.redis_client.incr('connections')
        self.redis_client.set('updated', time.time())

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.channel_layer.group_send(
            self.room_group_name,
            {
                'type': 'connection_message',
                'updated': ???,
                'connections': ???
            }
        )

        await self.accept()
```


```python
self.redis_client.incr('connections')
self.redis_client.set('updated', time.time())
```

```python
class CountConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = ???
        self.room_group_name = ???
        self.redis_client = redis.Redis(???)

        self.redis_client.incr('connections')
        self.redis_client.set('updated', time.time())

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.channel_layer.group_send(
            self.room_group_name,
            {
                'type': 'connection_message',
                'updated': ???,
                'connections': ???
            }
        )

        await self.accept()
```

```python
await self.channel_layer.group_send(
    self.room_group_name,
    {
        'type': 'connection_message',
        'updated': ???,
        'connections': ???
    }
)
```

```python
async def disconnect(self, close_code):
    self.redis_client = redis.Redis(???)
    self.redis_client.decr('connections')

    await self.channel_layer.group_discard(
        self.room_group_name,
        self.channel_name
    )

    await self.channel_layer.group_send(
        self.room_group_name,
        {
            'type': 'connection_message',
            'updated': ???,
            'connections': ???
        }
    )
```

```python
async def disconnect(self, close_code):
    self.redis_client = redis.Redis(???)
    self.redis_client.decr('connections')

    await self.channel_layer.group_discard(
        self.room_group_name,
        self.channel_name
    )

    await self.channel_layer.group_send(
        self.room_group_name,
        {
            'type': 'connection_message',
            'updated': ???,
            'connections': ???
        }
    )
```


```python
async def disconnect(self, close_code):
    self.redis_client = redis.Redis(???)
    self.redis_client.decr('connections')
```

```python
async def disconnect(self, close_code):
    self.redis_client = redis.Redis(???)
    self.redis_client.decr('connections')

    await self.channel_layer.group_discard(
        self.room_group_name,
        self.channel_name
    )

    await self.channel_layer.group_send(
        self.room_group_name,
        {
            'type': 'connection_message',
            'updated': ???,
            'connections': ???
        }
    )
```

```python
await self.channel_layer.group_send(
    self.room_group_name,
    {
        'type': 'connection_message',
        'updated': ???,
        'connections': ???
    }
)
```

```
<script>
  var chatSocket = new WebSocket(URL)

  chatSocket.onmessage = function(e) {
      var data = JSON.parse(e.data)
      var totalConnections = data['connections']
      var lastUpdated = data['updated']

      document.querySelector('#count').textContent = totalConnections
      document.querySelector('#updated').textContent = lastUpdated
  }

  chatSocket.onclose = function(e) {
      console.error('Chat socket closed unexpectedly');
  }

</script>
```

```
<script>
  var chatSocket = new WebSocket(URL)

  chatSocket.onmessage = function(e) {
      var data = JSON.parse(e.data)
      var totalConnections = data['connections']
      var lastUpdated = data['updated']

      document.querySelector('#count').textContent = totalConnections
      document.querySelector('#updated').textContent = lastUpdated
  }

  chatSocket.onclose = function(e) {
      console.error('Chat socket closed unexpectedly');
  }

</script>
```

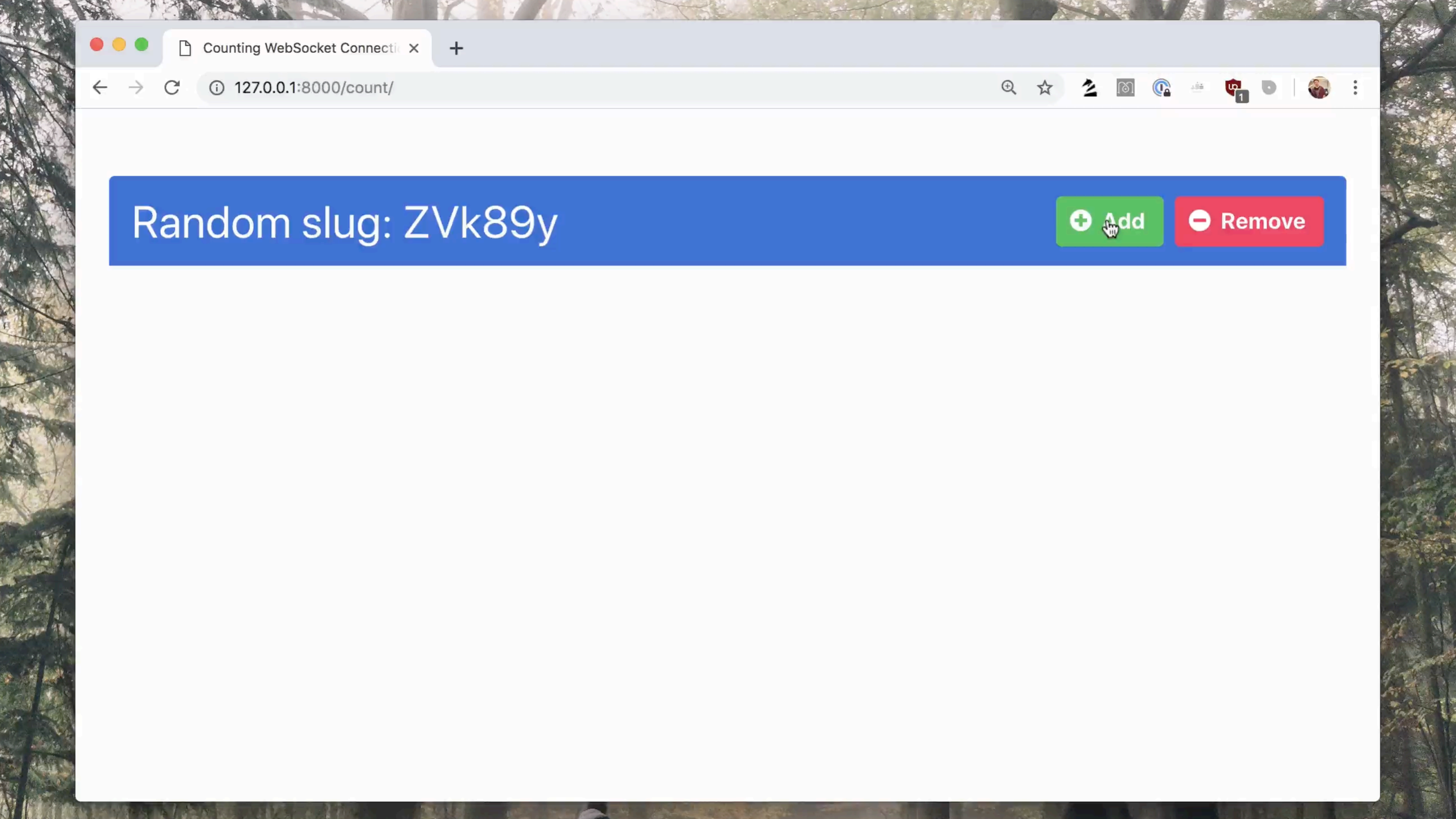
```
var chatSocket = new WebSocket(URL)
```

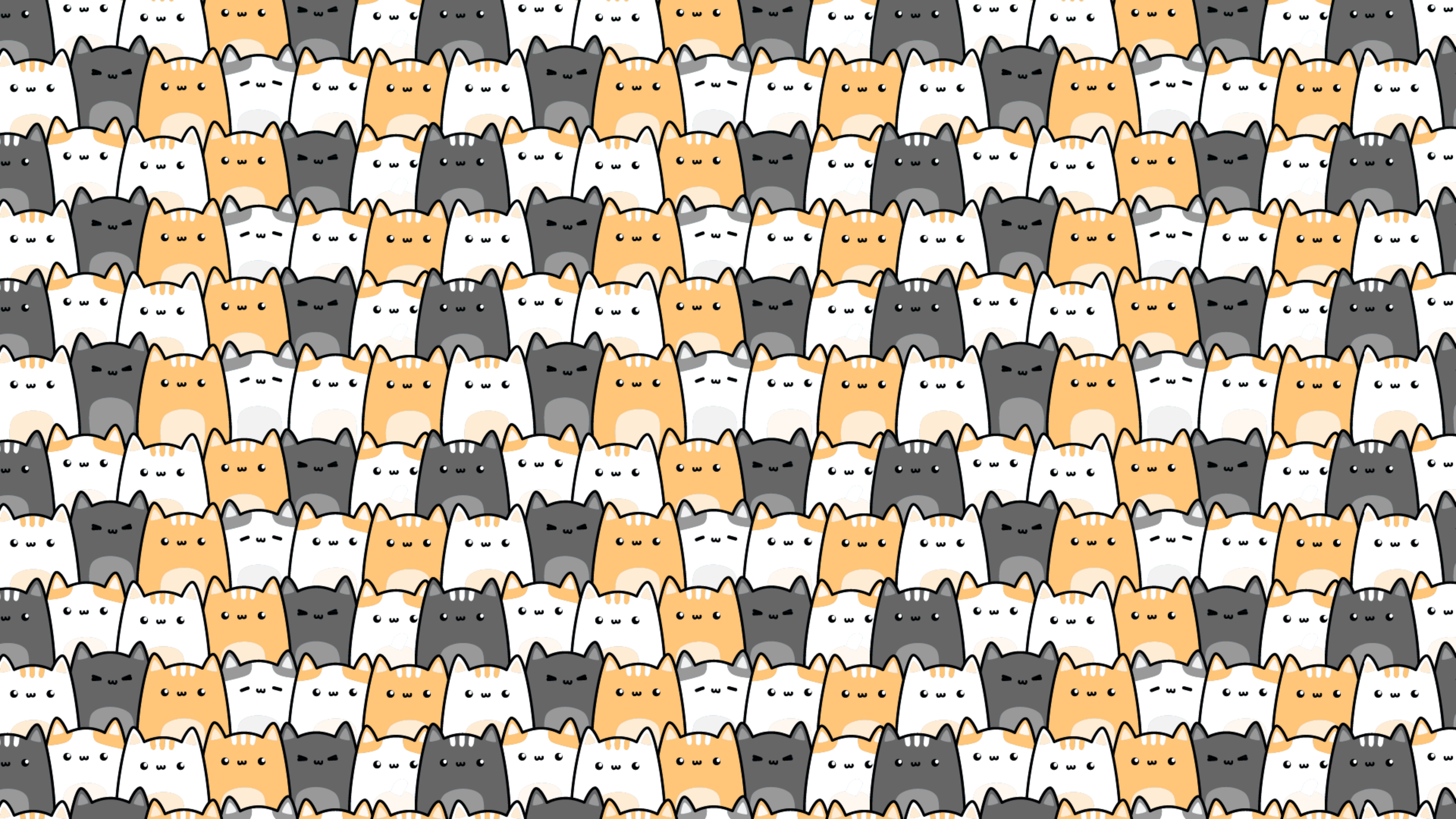
```
<script>
  var chatSocket = new WebSocket(URL)

  chatSocket.onmessage = function(e) {
      var data = JSON.parse(e.data)
      var totalConnections = data['connections']
      var lastUpdated = data['updated']

      document.querySelector('#count').textContent = totalConnections
      document.querySelector('#updated').textContent = lastUpdated
  }

  chatSocket.onclose = function(e) {
      console.error('Chat socket closed unexpectedly');
  }

</script>
```

```
chatSocket.onmessage = function(e) {
    var data = JSON.parse(e.data)
    var totalConnections = data['connections']
    var lastUpdated = data['updated']

    document.querySelector('#count').textContent = totalConnections
    document.querySelector('#updated').textContent = lastUpdated
}
```

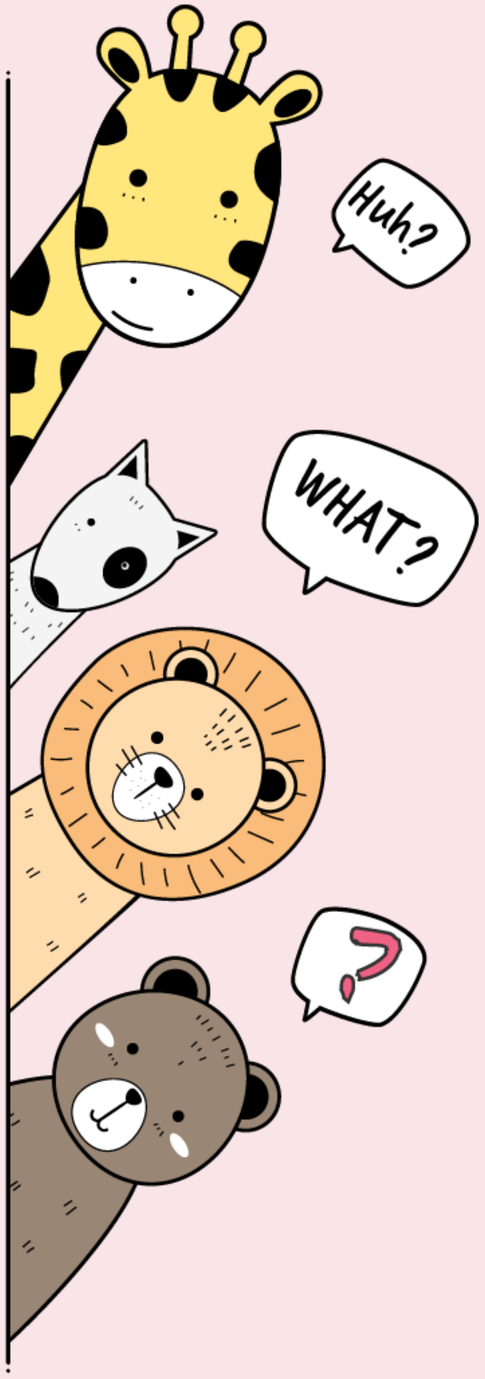
```
<script>
  var chatSocket = new WebSocket(URL)

  chatSocket.onmessage = function(e) {
      var data = JSON.parse(e.data)
      var totalConnections = data['connections']
      var lastUpdated = data['updated']

      document.querySelector('#count').textContent = totalConnections
      document.querySelector('#updated').textContent = lastUpdated
  }

  chatSocket.onclose = function(e) {
      console.error('Chat socket closed unexpectedly');
  }

</script>
```

```
chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
}
```


Counting WebSocket Connecti
127.0.0.1:8000/count/
Random slug: ZVk89y
Add
Remove



Shared WebWorkers!



nexmo® | The Vonage® API Platform

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```


```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
if (!sharedWebSocket) {
  sharedWebSocket = new WebSocket(URL);
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
sharedWebSocket.onmessage = function(e) {
  connections.forEach(function(connection) {
    let data = JSON.parse(e.data);
    connection.postMessage(data);
  });
};
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
sharedWebSocket.onmessage = function(e) {
  connections.forEach(function(connection) {
    let data = JSON.parse(e.data);
    connection.postMessage(data);
  });
};
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
sharedWebSocket.onmessage = function(e) {
  connections.forEach(function(connection) {
    let data = JSON.parse(e.data);
    connection.postMessage(data);
  });
};
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
sharedWebSocket.onmessage = function(e) {
  connections.forEach(function(connection) {
    let data = JSON.parse(e.data);
    connection.postMessage(data);
  });
};
```

```
const connections = [];
var sharedWebSocket;

onconnect = function(e) {
  const port = e.ports[0];
  connections.push(port);
  port.start();

  if (!sharedWebSocket) {
    sharedWebSocket = new WebSocket(URL);

    sharedWebSocket.onmessage = function(e) {
      connections.forEach(function(connection) {
        let data = JSON.parse(e.data);
        connection.postMessage(data);
      });
    };
  }

  sharedWebSocket.send("PING");
};
```

```
sharedWebSocket.onmessage = function(e) {
  connections.forEach(function(connection) {
    let data = JSON.parse(e.data);
    connection.postMessage(data);
  });
};
```

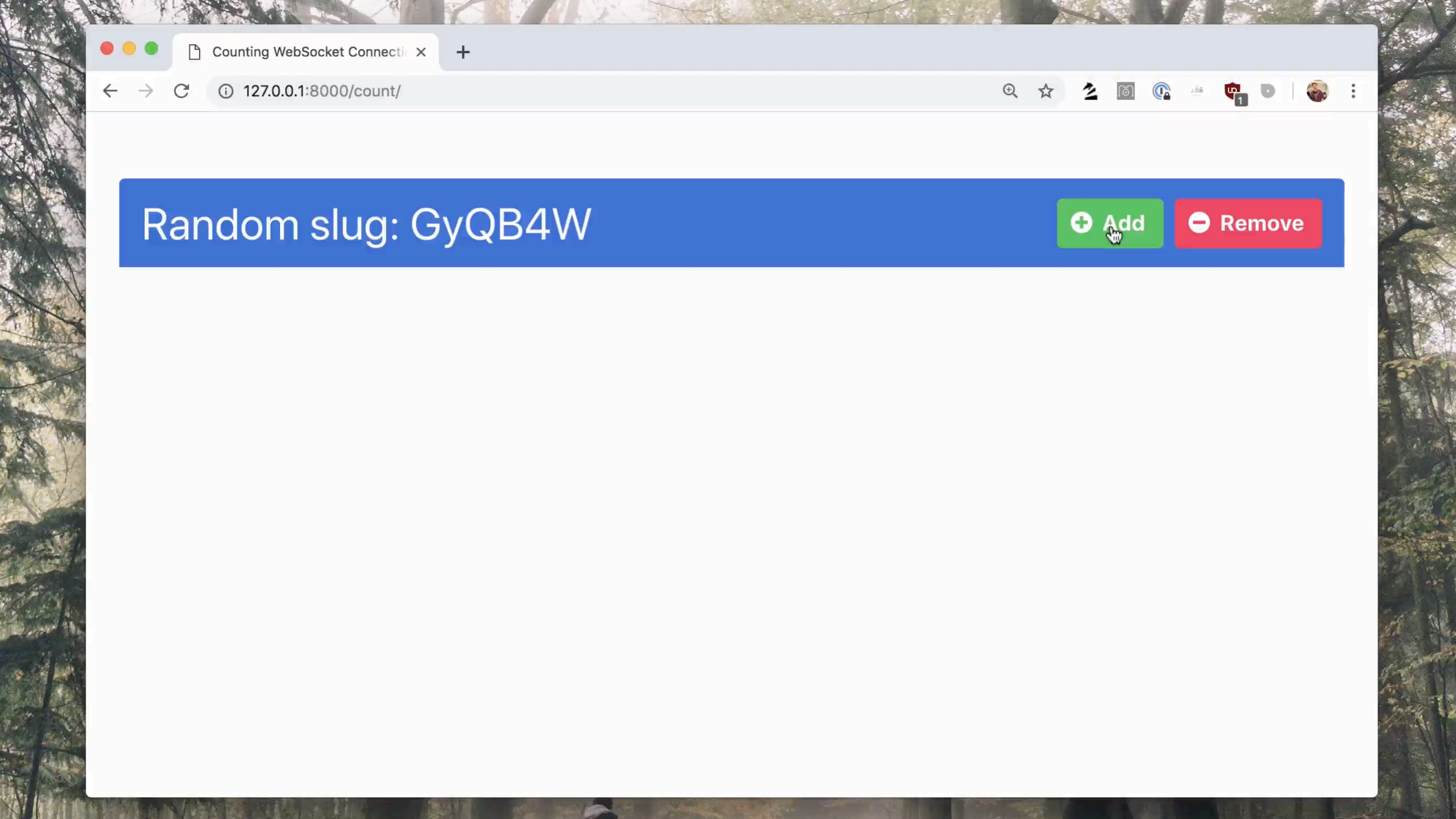
```
const worker = new SharedWorker('js/shared-worker.js')

worker.port.onmessage = function(e) {
  var data = e.data
  var totalConnections = data['connections']
  var lastUpdated = data['updated']

  document.querySelector('#count').textContent = totalConnections
  document.querySelector('#updated').textContent = lastUpdated
}

worker.port.start()
```

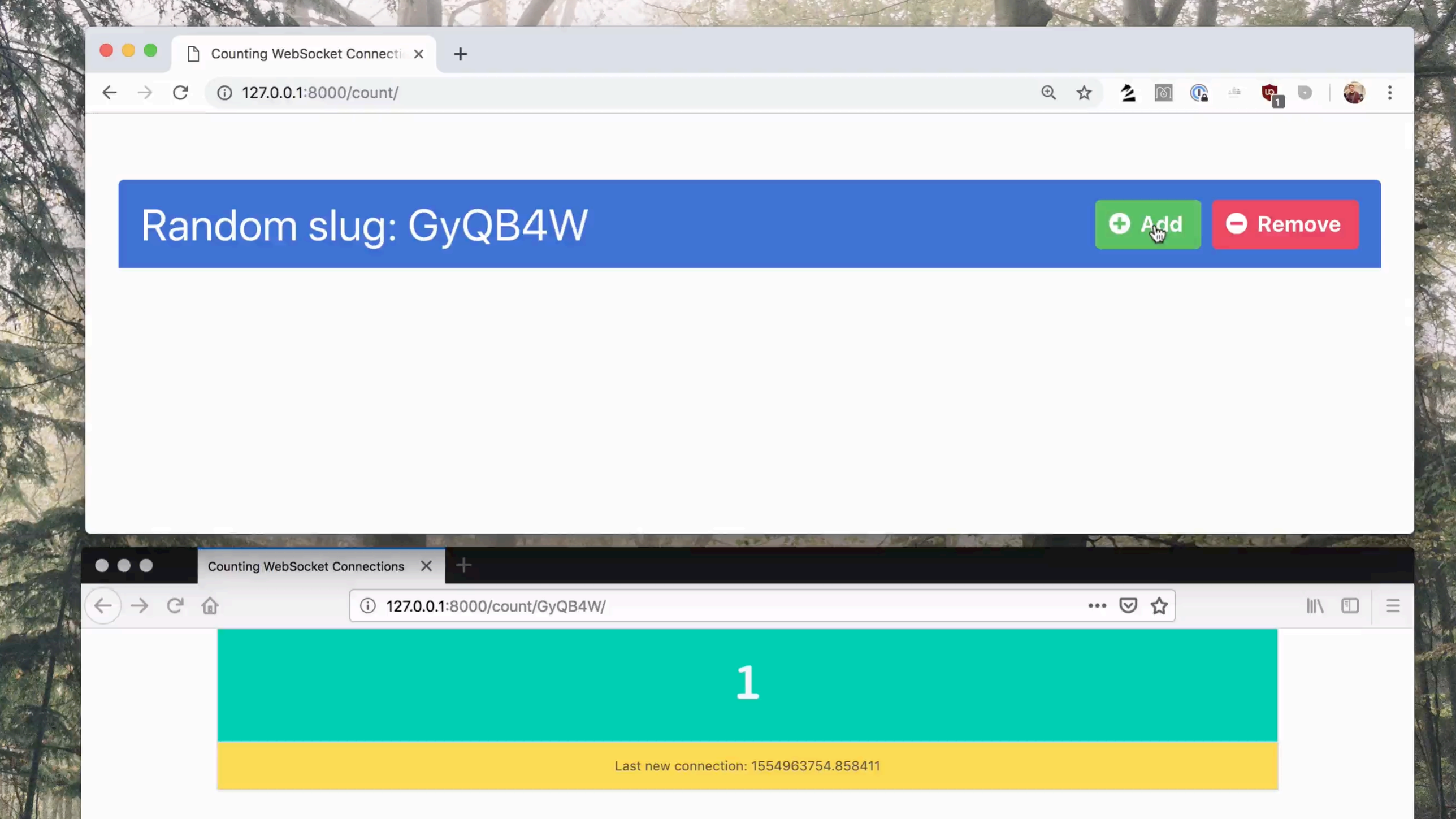
```
const worker = new SharedWorker('js/shared-worker.js')

worker.port.onmessage = function(e) {
  var data = e.data
  var totalConnections = data['connections']
  var lastUpdated = data['updated']

  document.querySelector('#count').textContent = totalConnections
  document.querySelector('#updated').textContent = lastUpdated
}

worker.port.start()
```

const worker = new SharedWorker('js/shared-worker.js')

```
const worker = new SharedWorker('js/shared-worker.js')

worker.port.onmessage = function(e) {
  var data = e.data
  var totalConnections = data['connections']
  var lastUpdated = data['updated']

  document.querySelector('#count').textContent = totalConnections
  document.querySelector('#updated').textContent = lastUpdated
}

worker.port.start()
```


```
worker.port.onmessage = function(e) {
  var data = e.data
  var totalConnections = data['connections']
  var lastUpdated = data['updated']

  document.querySelector('#count').textContent = totalConnections
  document.querySelector('#updated').textContent = lastUpdated
}
```


Counting WebSocket Connecti
127.0.0.1:8000/count/
Random slug: GyQB4W
Add
Remove


Counting WebSocket Connecti
127.0.0.1:8000/count/
Random slug: GyQB4W
Add
Remove
Counting WebSocket Connections
127.0.0.1:8000/count/GyQB4W/
1
Last new connection: 1554963754.858411

Shared Web Workers - LS

Usage % of all users ?

Global 42.1%

Method of allowing multiple scripts to communicate with a single web worker.

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE M
				3.1-4		3.2-4.3							
		2-28		5-6	10.1	5-6.1							
6-10	12-17	29-65	4-72	6.1-12	11.5-57	7-11.4		2.1-4.4.4	7	12-12.1			1
11	18	66	73	12.1	58	12.1	all	67	10	46	71	64	1
		67-68	74-76	TP		12.2							

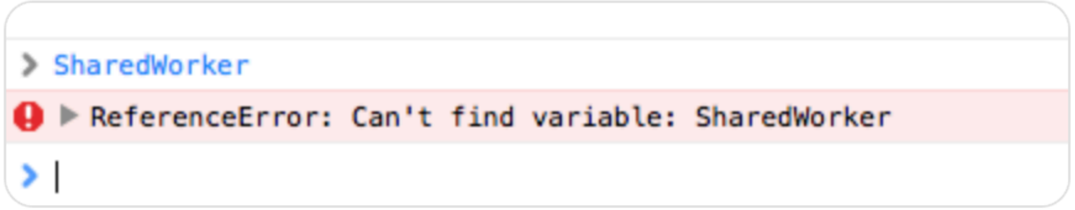
Notes | Known issues (0) | Resources (6) | Feedback

MS Edge status: Not currently planned

WebKit status: Removed

tregoning @tregoning · 6 Oct 2015

@xeenon do you think SharedWorkers might come back to WebKit at some point win the future?



1

Timothy Hatcher

@xeenon

Follow

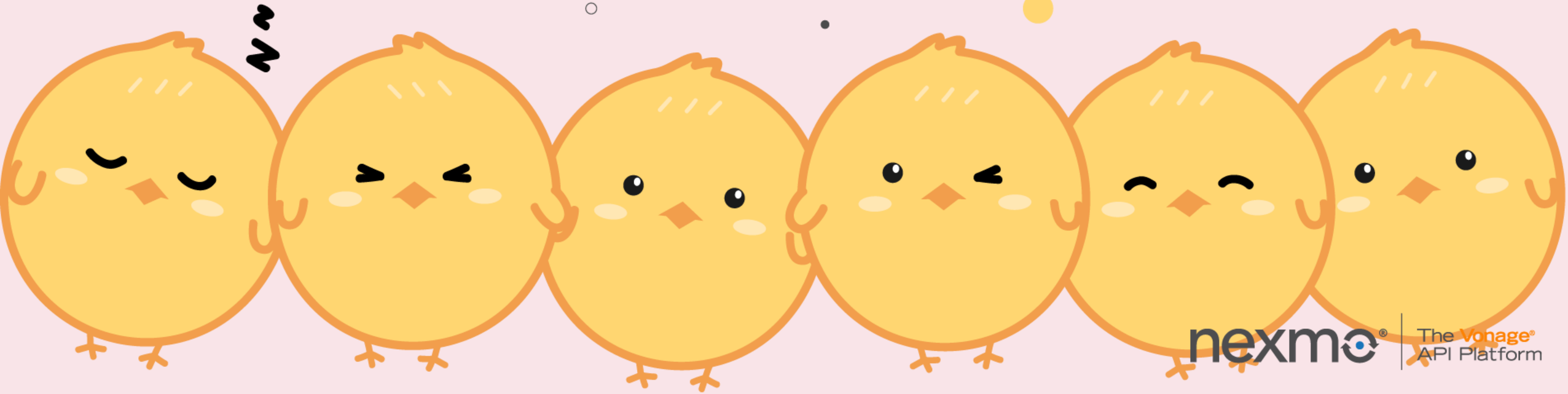
Replying to @tregoning

@tregoning The implementation of Shared Web Workers was imposing undesirable constraints on the engine. It never gained any adoption.

8:55 PM - 9 Oct 2015

1

@aaronbassett



nexmo® | The Vonage® API Platform