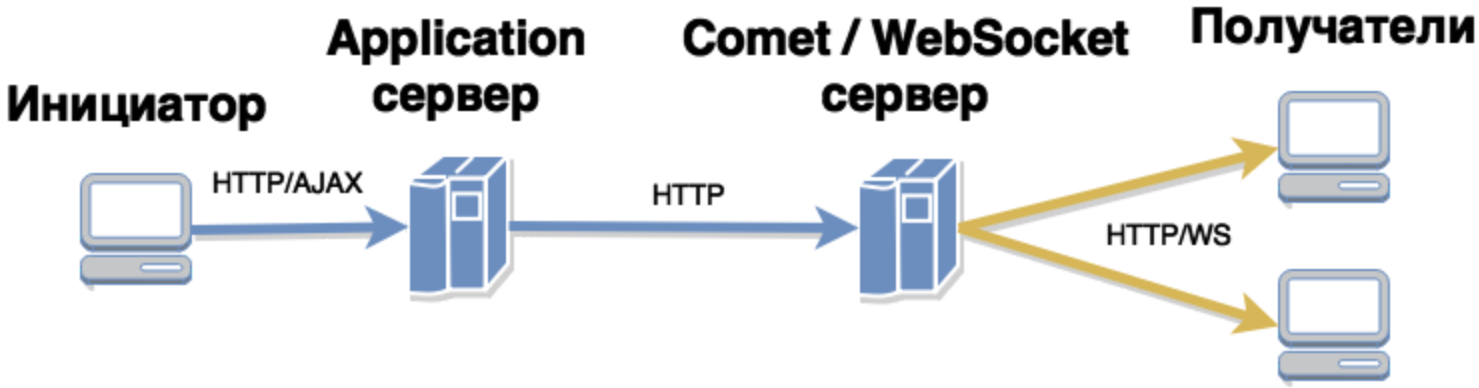
Real Time сообщения

Примеры использования

- Чаты и мессенджеры
- Отображение котировок
- Прямые трансляции (a-la twitter)
- Push уведомления
- Сетевой обмен в играх на HTML

Архитектура



Решения

- **Polling** - периодический опрос сервера
- **Comet (Long polling)** - polling с долгоживущими запросами
- **WebSocket** - специализированный протокол

Polling

Polling - периодический опрос

Polling на клиенте

```
var since = 0;
setInterval(function() {
    $.ajax({
        type: 'GET',
        url:  '/messages/',
        data: { cid: 5, since: since },
    }).success(function(resp) {
        if (!resp.messages || !resp.messages.length) {
            return;
        }
        handleMessages(resp.messages);
        since = resp.messages[0].id;
    });
}, 5000);
```

Polling на сервере

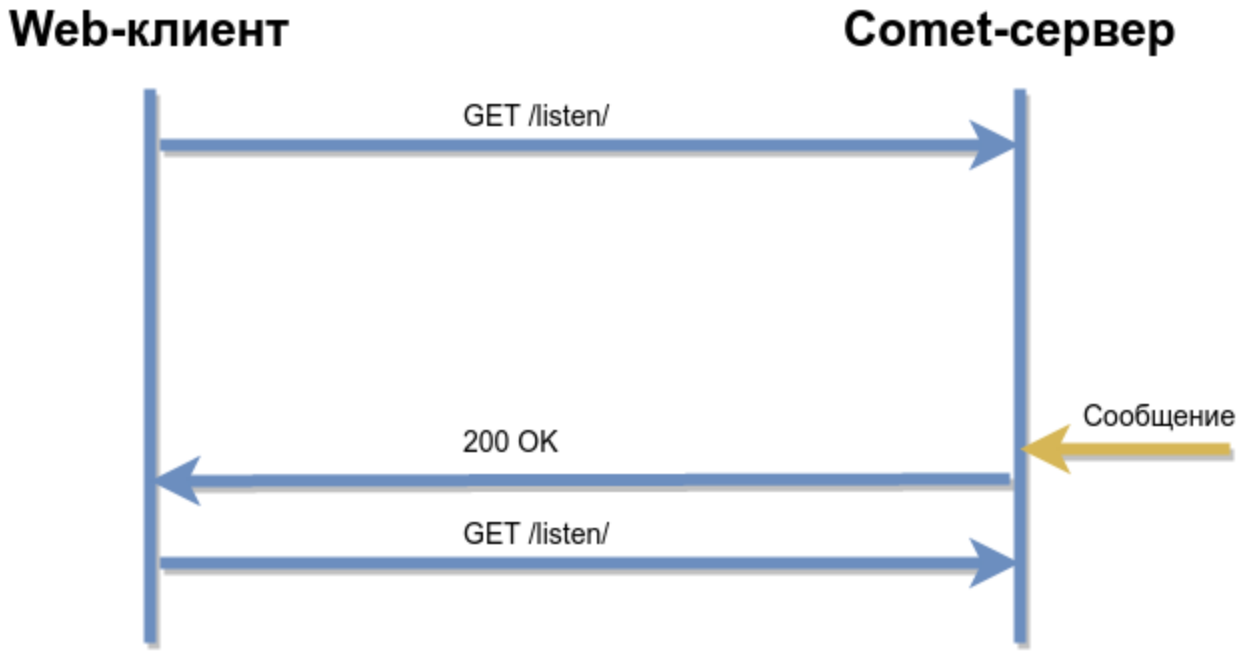
```
def messages(request):
    cid = request.GET.get("cid")
    since = request.GET.get("since", 0)
    messages = Messages.objects.filter(
        cid = cid,
        id__gt = since,
    ).order_by("-id")
    messages = [ m.as_data() for m in messages ]
    return HttpResponseAjax(messages = messages)
```

Плюсы и минусы Polling

- \+ Простота и надежность реализации
- \+ Не требуется дополнительного ПО
- − Сообщения приходят с задержкой до N секунд
- − Избыточное число HTTP запросов `RPS=CCU/N`
- − Ограничение по числу пользователей

Comet

Comet - долгоживущие запросы



Comet на клиенте

```
function getComet() {
    $.ajax({
        type: 'GET',
        url:  '/listen/',
        data: { cid: 5 },
    }).success(function(resp) {
        handleMessages(resp.messages);
        getComet();
    }).error(function() {
        setTimeout(getComet, 10000);
    });
}
getComet();
```

Comet на сервере

В технологии **comet** сервер должен поддерживать одновременно открытыми большое количество соединений, причем каждое соединение находится в ожидании сообщений для него. По этой причине мы не можем использовать классический application-сервер в роли comet-сервера. Для comet-сервера необходима отдельная технология, например **nginx + push-stream-module**.

Nginx + push-stream-module

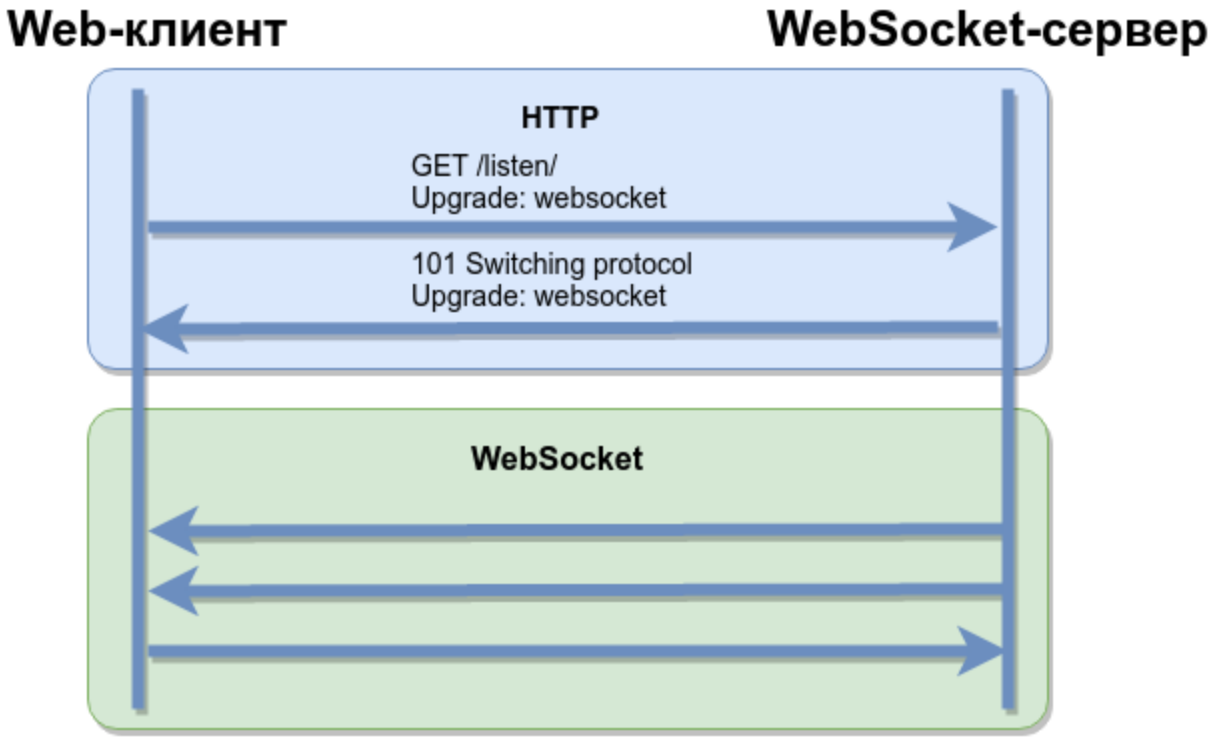
```
location /publish/ {
    push_stream_publisher normal;        # включаем отправку
    push_stream_channels_path $arg_cid; # id канала
    push_stream_store_messages off;      # не храним сообщения
    allow   127.0.0.1;
    deny    all;
}

location /listen/ {
    push_stream_subscriber long-polling;    # включаем доставку
    push_stream_channels_path  $arg_cid;    # id канала
    default_type application/json;          # MIME тип сообщения
}
```

Плюсы и минусы Comet

- \+ Поддержка всеми браузерами
- \+ Поддержка большого числа пользователей
- \+ Относительная простота реализации
- − Избыточные HTTP запросы
- − Half-duplex

WebSocket

WebSocket


Web-клиент
WebSocket-сервер
HTTP
GET /listen/
Upgrade: websocket
101 Switching protocol
Upgrade: websocket
WebSocket

WebSocket handshake

```
GET /listen HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

WebSocket на стороне клиента

```
var socket = new WebSocket('ws://host/listen');

socket.onopen = function(event) {
    console.log('ws opened');
    var data = JSON.stringify({ message: "Hello WebSocket" });
    socket.send(data);
};
socket.onmessage = function(event) {
    var resp = JSON.parse(event.data);
    console.log('ws message', resp.message);
};
socket.onclose = function(event) {
    console.log('ws closed')
};
```

WebSocket на стороне сервера

```
registry = {}

class WSHandler(tornado.websocket.WebSocketHandler):
    def open(self):
        self.uid = self.get_argument("uid")
        registry[self.uid] = self
    def check_origin(self, origin):
        return True
    def on_close(self):
        del registry[self.uid]
```

WebSocket на стороне сервера (2)

```
class MainHandler(tornado.web.RequestHandler):
    def post(self):
        body = self.get_argument("msg")
        uid = self.get_argument("uid")
        conn = registry.get(uid)
        if conn:
            conn.write_message(body)
            self.write("OK")
        else:
            self.write("NO")
```

WebSocket на стороне сервера (3)

```
if __name__ == "__main__":
    app = tornado.web.Application([
        (r"/publish", MainHandler),
        (r"/listen", WSHandler),
    ])
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Плюсы и минусы WebSocket

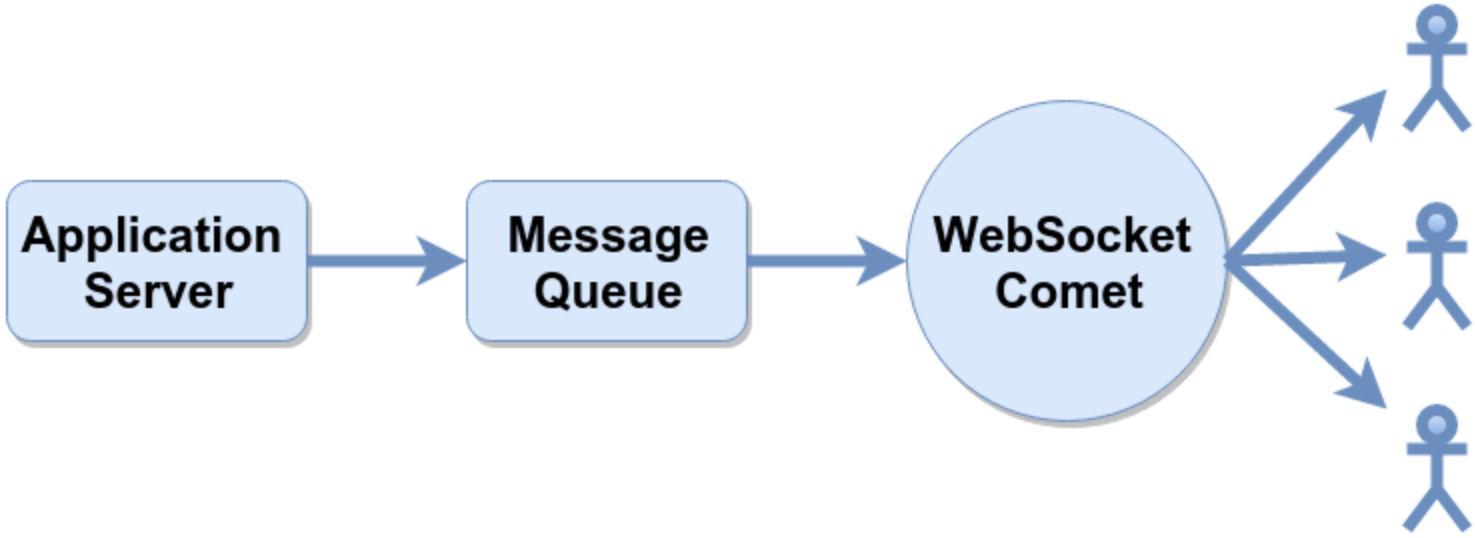
+ Минимальный объем трафика
+ Минимальная нагрузка на сервер
+ Поддержка большого числа пользователей
+ Full-duplex
− Нет поддержки IE<10, OperaMini, Android<4.4
− Требуется специальный WebSocket-сервер
− Плохо работает с прокси-серверами

Отправка сообщений

Отправка сообщений

```
import requests # pip install requests
import json
puburl = "http://127.0.0.1/publish/"
def send_message(request):
    cid = request.GET.get("to")
    text = request.GET.get("text")
    body = json.dumps({ "messages": [ text ] })
    try: ## может быть долгим
        resp = requests.post(puburl, params={"cid":cid}, data=body)
        if resp.status_code == 200:
            return HttpResponseAjax()
        else:
            return HttpResponseAjaxError(code=resp.status_code)
    except:
        return HttpResponseAjaxError(code=500)
```

Отправка через очередь



Софт для Real Time сообщений

- Real Time Web Technologies Guide - https://www.leggetter.co.uk/real-time-web-technologies-guide/
- Real Time libraries and frameworks - https://deepstream.io/blog/realtime-framework-overview/
- Centrifugo - https://github.com/centrifugal/centrifugo