Настройка балансировщика на базе HAProxy

- Прежде, чем начать
- Настройка WCS серверов
  - 1. Порты для приема входящих соединений
  - 2. Настройка WCS
  - 3. Настройка агента для HAProxy
- Настройка балансировщика
  - 1. Настройка nginx для раздачи примеров
  - 2. Настройка SSL сертификатов для HAProxy
  - 3. Настройка HAProxy
- Тестирование

HAProxy - это надежный инструмент для создания обратных прокси-серверов и балансировки нагрузки с открытым исходным кодом. В частности, его модифицированные сборки лежат в основе большинства известных балансировщиков, например, AWS LB. Рассмотрим, как настроить собственный балансировщик нагрузки при помощи HAProxy.

Прежде, чем начать

Для развертывания балансировщика нагрузки потребуются:

- серверы с установленным WCS (облачные или физические)
- отдельный сервер, который будет принимать входящие соединения от клиентов
- доменное имя и SSL-сертификат

Если WCS серверы должны входить в состав CDN, на них должна быть выполнена настройка CDN. Если необходимо балансировать публикации на несколько Origin инстансов, или проигрывание с нескольких Edge инстансов, эти инстансы должны быть настроены заранее.

Настройка WCS серверов

1. Порты для приема входящих соединений

Открываем необходимые порты для входящих соединений на каждом из WCS серверов (если это не сделано ранее). Пример минимального набора портов из настройки инстансов в AWS EC2

Port range	Protocol	Source
22	TCP	0.0.0.0/0
8080 - 8084	TCP	0.0.0.0/0
443	TCP	0.0.0.0/0
8888	TCP	0.0.0.0/0
30000 - 33000	UDP	0.0.0.0/0
8443 - 8445	TCP	0.0.0.0/0
1935	UDP	0.0.0.0/0
9091	TCP	0.0.0.0/0
80	TCP	0.0.0.0/0
9707	TCP	0.0.0.0/0

Обратите внимание, что к обычному набору портов добавляется TCP порт 9707. Этот порт HAProxy будет использовать для контроля текущего состояния сервера.

Порты для передачи медиатрафика (30000-33000 в примере выше) должны быть доступны извне в случае, если сервер располагается за NAT, поскольку HAProxy может проксировать только Websocket соединения, но не WebRTC.

2. Настройка WCS

Добавляем в файл `flashphoner.properties` настройки для использования реальных IP адресов клиентов в идентификаторах сессии

```
ws.map_custom_headers=true
ws.ip_forward_header=X-Client-IP
```

Если планируется распределять нагрузку между серверами в зависимости от пропускной способности канала, добавляем также настройку

```
global_bandwidth_check_enabled=true
```

После этого перезапускаем WCS

```
sudo systemctl restart webcallserver
```

3. Настройка агента для HAProxy

3.1. Устанавливаем необходимые зависимости на сервер

```
yum install jq bc xinetd telnet
```

3.2. Копируем скрипты `haproxy-agent-check.sh` и `haproxy-agent-check-launch.sh` в каталог `/usr/local/bin` и даем права на исполнение

```
sudo cp haproxy-agent-check* /usr/local/bin/
sudo chmod +x /usr/local/bin/haproxy-agent-check*
```

haproxy-agent-check-launch.sh

```
#! /bin/bash
/usr/local/bin/haproxy-agent-check.sh cpu 70
```

haproxy-agent-check.sh

```
#!/bin/bash
CPU_MAX_LOAD=90
MAX_PUBLISHERS=100
MAX_SUBSCRIBERS=100
MAX_HLS_STREAMS=100
MAX_BANDWIDTH_IN=100
MAX_BANDWIDTH_OUT=100

function isTreshold_Cpu() {
    local load=$(uptime | grep -E -o 'load average[s:][: ].*' | sed 's/,//g' | cut -d' ' -f3-5)
    local cpus=$(grep processor /proc/cpuinfo | wc -l)
    local l5util=0

    while read -r l1 l5 l15; do {
        l5util=$(echo "pct=$l5/$cpus*100; if(pct<1) print 0; pct" | bc -l | cut -d"." -f1);
        if [[ $l5util -lt $CPU_MAX_LOAD ]]; then
            true; return
        else
            false; return
        fi
    }; done < <(echo $load)
}

function isTreshold_Publishers() {
    local statsJson=$1
```

```
    local webrtcPublishers=$(echo $statsJson | jq '.streams_stats.streams_webrtc_in' | bc -l)
    local rtmpPublishers=$(echo $statsJson | jq '.streams_stats.streams_rtmp_in' | bc -l)
    local rtspStreamsIn=$(echo $statsJson | jq '.streams_stats.streams_rtsp_in' | bc -l)
    local rtspPublishers=$(echo $statsJson | jq '.streams_stats.streams_rtsp_push_in' | bc -l)

    local publishers=$(($webrtcPublishers + $rtmpPublishers + $rtspStreamsIn + $rtspPublishers))
    if [[ $publishers -lt $MAX_PUBLISHERS ]]; then
        true; return
    else
        false; return
    fi
}

function isTreshold_Subscribers() {
    local statsJson=$1
    local webrtcSubscribers=$(echo $statsJson | jq '.streams_stats.streams_webrtc_out' | bc -l)
    local rtmpSubscribers=$(echo $statsJson | jq '.streams_stats.streams_rtmp_out' | bc -l)
    local rtmpRepublishers=$(echo $statsJson | jq '.streams_stats.streams_rtmp_client_out' | bc -l)
    local rtspSubscribers=$(echo $statsJson | jq '.streams_stats.streams_rtsp_out' | bc -l)
    local websocketSubscribers=$(echo $statsJson | jq '.streams_stats.streams_websocket_out' | bc -l)

    local subscribers=$(($webrtcSubscribers + $rtmpSubscribers + $rtmpRepublishers + $rtspSubscribers +
$websocketSubscribers))
    if [[ $subscribers -lt $MAX_SUBSCRIBERS ]]; then
        true; return
    else
        false; return
    fi
}

function isTreshold_HlsStreams() {
    local statsJson=$1
    local hlsStreams=$(echo $statsJson | jq '.streams_stats.streams_hls' | bc -l)
    if [[ $hlsStreams -lt $MAX_HLS_STREAMS ]]; then
        true; return
    else
        false; return
    fi
}

function isTreshold_BandwidthIn() {
    local statsJson=$1
    local bandwidthIn=$(echo $statsJson | jq '.network_stats.global_bandwidth_in' | bc -l)
    local comparison=$(echo "$bandwidthIn < $MAX_BANDWIDTH_IN" | bc -l)
    if [[ $comparison -ne 0 ]]; then
        true; return
    else
        false; return
    fi
}

function isTreshold_BandwidthOut() {
    local statsJson=$1
    local bandwidthOut=$(echo $statsJson | jq '.network_stats.global_bandwidth_out' | bc -l)
    local comparison=$(echo "$bandwidthOut < $MAX_BANDWIDTH_OUT" | bc -l)
    if [[ $comparison -ne 0  ]]; then
        true; return
    else
        false; return
    fi
}

function usage() {
    echo "Usage: $(basename $0) [OPTIONS]"
    echo -e "  cpu [<treshold>]\t\tcheck CPU load (default 90 %)"
    echo -e "  publishers [<treshold>]\tcheck publishers count (default 100)"
    echo -e "  subscribers [<treshold>]\tcheck subscribers count (default 100)"
    echo -e "  hls [<treshold>]\t\tcheck HLS streams count (default 100)"
    echo -e "  band-in [<treshold>]\t\tcheck incoming channel bandwidth (default 100 Mbps)"
    echo -e "  band-out [<treshold>]\t\tcheck outgoing channel bandwidth (default 100 Mbps)"
    echo ""
```

```
    echo -e "Example: $(basename $0) cpu 90 publishers 100 subscribers 100 hls 100 band-in 100 band-out 100"
    exit 0
}


function main() {
    local checklist=()
    local statsJson=""
    local check=""

    if [[ $# -eq 0 ]]; then
        checklist=(
            'Cpu'
            'Publishers'
            'Subscribers'
            'HlsStreams'
            'BandwidthIn'
            'BandwidthOut'
        )
    else
        while [[ $# -gt 0 ]]; do
            case $1 in
                cpu)
                    checklist+=('Cpu')
                    if [ -z "${2//[0-9]}" ]; then
                        CPU_MAX_LOAD=$2
                        shift
                    fi
                    shift
                    ;;
                publishers)
                    checklist+=('Publishers')
                    if [ -z "${2//[0-9]}" ]; then
                        MAX_PUBLISHERS=$2
                        shift
                    fi
                    shift
                    ;;
                subscribers)
                    checklist+=('Subscribers')
                    if [ -z "${2//[0-9]}" ]; then
                        MAX_SUBSCRIBERS=$2
                        shift
                    fi
                    shift
                    ;;
                hls)
                    checklist+=('HlsStreams')
                    if [ -z "${2//[0-9]}" ]; then
                        MAX_HLS_STREAMS=$2
                        shift
                    fi
                    shift
                    ;;
                band-in)
                    checklist+=('BandwidthIn')
                    if [ -z "${2//[0-9]}" ]; then
                        MAX_BANDWIDTH_IN=$2
                        shift
                    fi
                    shift
                    ;;
                band-out)
                    checklist+=('BandwidthOut')
                    if [ -z "${2//[0-9]}" ]; then
                        MAX_BANDWIDTH_OUT=$2
                        shift
                    fi
                    shift
                    ;;
                help|*)
```

```
                    usage
                    ;;
            esac
        done
    fi
    if [[ -z "${checklist[@]}" ]]; then
        usage
        return 1
    fi

    statsJson=$(curl -s 'http://localhost:8081/?action=stat&format=json')
    if [[ -z "$statsJson" ]]; then
        echo "down"
        return 1
    fi
    for check in ${checklist[@]}; do
        if ! isTreshold_$check $statsJson; then
            echo "down"
            return 1
        fi
    done
    echo "up 100%"
    return 0
}

main "$@"

exit $?
```

Скрипт `haproxy-agent-check.sh` используется для получения состояния сервера на основе системной информации и статистики работы WCS. Скрипту указываются параметры, при превышении которых он возвращает значение `down` . В свою очередь, HAProxy, получив это значение, прекращает передачу новых входящих соединений на этот сервер до тех пор, пока не получит от скрипта значение `up` .

Поддерживаются следующие граничные условия:

- cpu - максимальная средняя загрузка CPU, в процентах, по умолчанию 90
- publishers - максимальное количество публикаций на сервере, включая WebRTC, RTMP, RTSP потоки, по умолчанию 100
- subscribers - максимальное количество подписчиков на сервере, включая WebRTC, RTMP, RTSP подписчиков, по умолчанию 100
- hls - максимальное количество HLS потоков на сервере, по умолчанию 100
- band-in - максимальная нагрузка на входящий канал, по умолчанию 100 Мбит/с
- band-out - максимальная нагрузка на входящий канал, по умолчанию 100 Мбит/с

Например, для контроля загрузки CPU не выше 70% скрипт должен быть вызван с параметром

```
/usr/local/bin/haproxy-agent-check.sh cpu 70
```

3.3. Добавляем в файл `/etc/services` строку

```
haproxy-agent-check 9707/tcp # haproxy-agent-check
```

3.4. В каталог `/etc/xinetd.d` добавляем файл `haproxy-agent-check`

```
# default: on
# description: haproxy-agent-check
service haproxy-agent-check
{
        disable         = no
        flags           = REUSE
        socket_type     = stream
        port            = 9707
        wait            = no
        user            = nobody
        server          = /usr/local/bin/haproxy-agent-check-launch.sh
        log_on_failure  += USERID
        only_from       = 172.31.42.154 127.0.0.1
        per_source      = UNLIMITED
}
```

Скрипт `haproxy-agent-check-launch.sh` используется, поскольку xinetd не поддерживает указание параметров командной строки в параметре `server` .

Параметр `only_from` разрешает соединения к порту 9707 только с сервера, где будет установлен HAProxy, а также локальные соединения для тестирования.

3.5. Даем файлу `haproxy-agent-check` права на исполнение

```
sudo chmod +x /etc/xinetd/haproxy-agent-check
```

3.6. Перезапускаем xinetd

```
sudo systemctl restart xinetd
```

3.7. Проверяем работу агента

```
telnet localhost 9707
```

```
[ec2-user@ip-172-31-44-243 ~]$ telnet localhost 9707
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
up 100%
Connection closed by foreign host.
```

Настройка балансировщика

1. Настройка nginx для раздачи примеров

1.1. Устанавливаем nginx

```
sudo yum install nginx
```

1.2. В файле `/etc/nginx/nginx.conf` меняем порт по умолчанию, а также имя сервера на `localhost`

```
    server {
        listen       8180;
        listen       [::]:8180;
        server_name  localhost;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        error_page 404 /404.html;
        location = /404.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

nginx будет доступен только локально, точку входя для клиентов будет обслуживать HAProxy.

1.3. Перезапускаем nginx

```
sudo systemctl restart nginx
```

1.4. Загружаем актуальную сборку WebSDK

```
wget https://flashphoner.com/downloads/builds/flashphoner_client/wcs_api-2.0/flashphoner-api-2.0.206-
7d9863ae4de631a59ff8793ddecd104ca2fd4a22.tar.gz
```

и распаковываем ее в каталог `/usr/share/nginx/html/wcs`

```
sudo mkdir /usr/share/nginx/html/wcs
cd /usr/share/nginx/html/wcs
sudo tar -xzf ~/flashphoner-api-2.0.206-7d9863ae4de631a59ff8793ddecd104ca2fd4a22.tar.gz --strip-components=2
```

2. Настройка SSL сертификатов для HAProxy

2.1. Создаем файл сертификата в PEM формате (должен включать все сертификаты и приватный ключ) и копируем в каталог. где файл сертификата будет постоянно доступен

```
cat cert.crt ca.crt cert.key >> cert.pem
sudo mkdir -p /etc/pki/tls/mydomain.com
sudo cp cert.pem /etc/pki/tls/mydomain.com
```

3. Настройка HAProxy

3.1. Устанавливаем HAProxy

```
sudo yum install haproxy
```

3.2. Редактируем файл `/etc/haproxy/haproxy.cfg`

haproxy.cfg example

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         /dev/log local0

    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000

#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend wcs-balancer
    bind *:443 ssl crt /etc/pki/tls/mydomain.com/cert.pem
    acl is_websocket hdr(Upgrade) -i WebSocket
    acl is_websocket hdr(Sec-WebSocket-Key) -m found
    use_backend wcs_back if is_websocket
    default_backend wcs_web_admin

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend wcs_back
    http-request add-header X-Client-IP %ci:%cp
    balance     roundrobin
    server wcs1_ws 172.31.44.243:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
    server wcs2_ws 172.31.33.112:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707

#---------------------------------------------------------------------
# WCS web admin dashboard
#---------------------------------------------------------------------
backend wcs_web_admin
    server wcs_web_http localhost:8180 maxconn 100 check
```

Параметры в секциях `global` и `defaults` можно оставить по умолчанию. Настраиваем точку входа

```
frontend wcs-balancer
    bind *:443 ssl crt /etc/pki/tls/mydomain.com/cert.pem
    acl is_websocket hdr(Upgrade) -i WebSocket
    acl is_websocket hdr(Sec-WebSocket-Key) -m found
    use_backend wcs_back if is_websocket
    default_backend wcs_web_admin
```

Бэкендом по умолчанию будет nginx с примерами из WebSDK

```
backend wcs_web_admin
    server wcs_web_http localhost:8180 maxconn 100 check
```

Бэкенд, балансирующий нагрузку между двумя инстансами (IP адреса приведены для примера)

```
backend wcs_back
    http-request add-header X-Client-IP %ci:%cp
    balance     roundrobin
    server wcs1_ws 172.31.44.243:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
    server wcs2_ws 172.31.33.112:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
```

```
#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend wcs-balancer
    bind *:443 ssl crt /etc/pki/tls/fpntest.com/cert.pem
    acl is_websocket hdr(Upgrade) -i WebSocket
    acl is_websocket hdr(Sec-WebSocket-Key) -m found
    use_backend wcs_back if is_websocket
    default_backend wcs_web_admin

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend wcs_back
    http-request add-header X-Client-IP %ci:%cp
    balance     roundrobin
    server wcs1_ws 172.31.44.243:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
    server wcs2_ws 172.31.33.112:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707

#---------------------------------------------------------------------
# WCS web admin dashboard
#---------------------------------------------------------------------
backend wcs_web_admin
    server wcs_web_http localhost:8180 maxconn 100 check
```

При необходимости, можно настроить залипание сессий

```
backend wcs_back
    http-request add-header X-Client-IP %ci:%cp
    balance     roundrobin
    cookie SERVERID insert indirect nocache
    server wcs1_ws 172.31.44.243:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
cookie wcs1_ws
    server wcs2_ws 172.31.33.112:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
cookie wcs1_ws
```

В этом случае соединения от одного и того же клиента будут направляться на один и тот же сервер, если только он не возвращает состояние d own

Также можно настроить балансировку по количеству соединений

```
backend wcs_back
    http-request add-header X-Client-IP %ci:%cp
    balance     leastconn
    server wcs1_ws 172.31.44.243:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
    server wcs2_ws 172.31.33.112:8080 maxconn 100 weight 100 check agent-check agent-inter 5s agent-port 9707
```

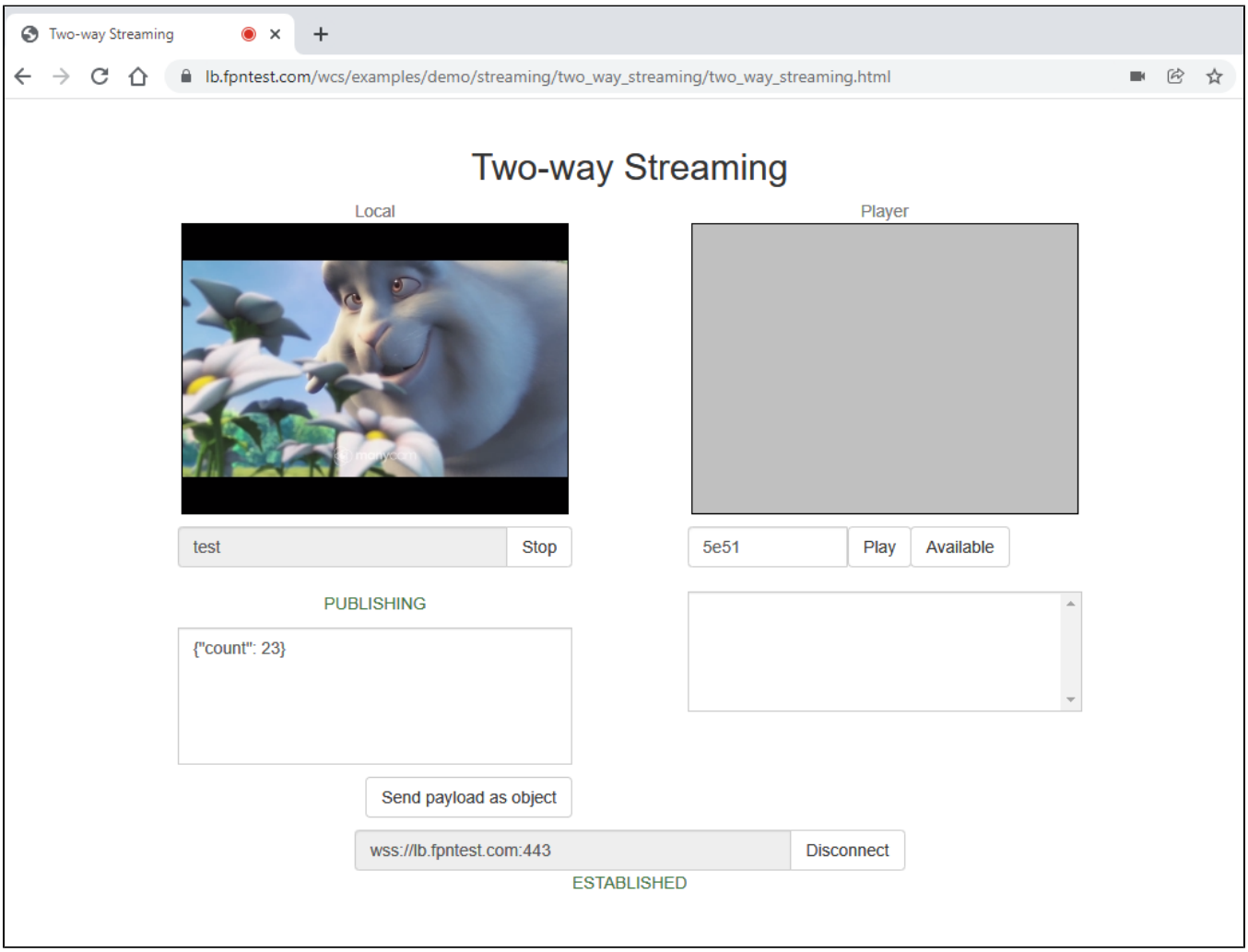
В этом случае клиенты будут направляться на первый сервер, пока указанное число соединений `maxconn` не будет достигнуто, либо пока сервер не вернет состояние `down`

3.3. Перезапускаем HAProxy

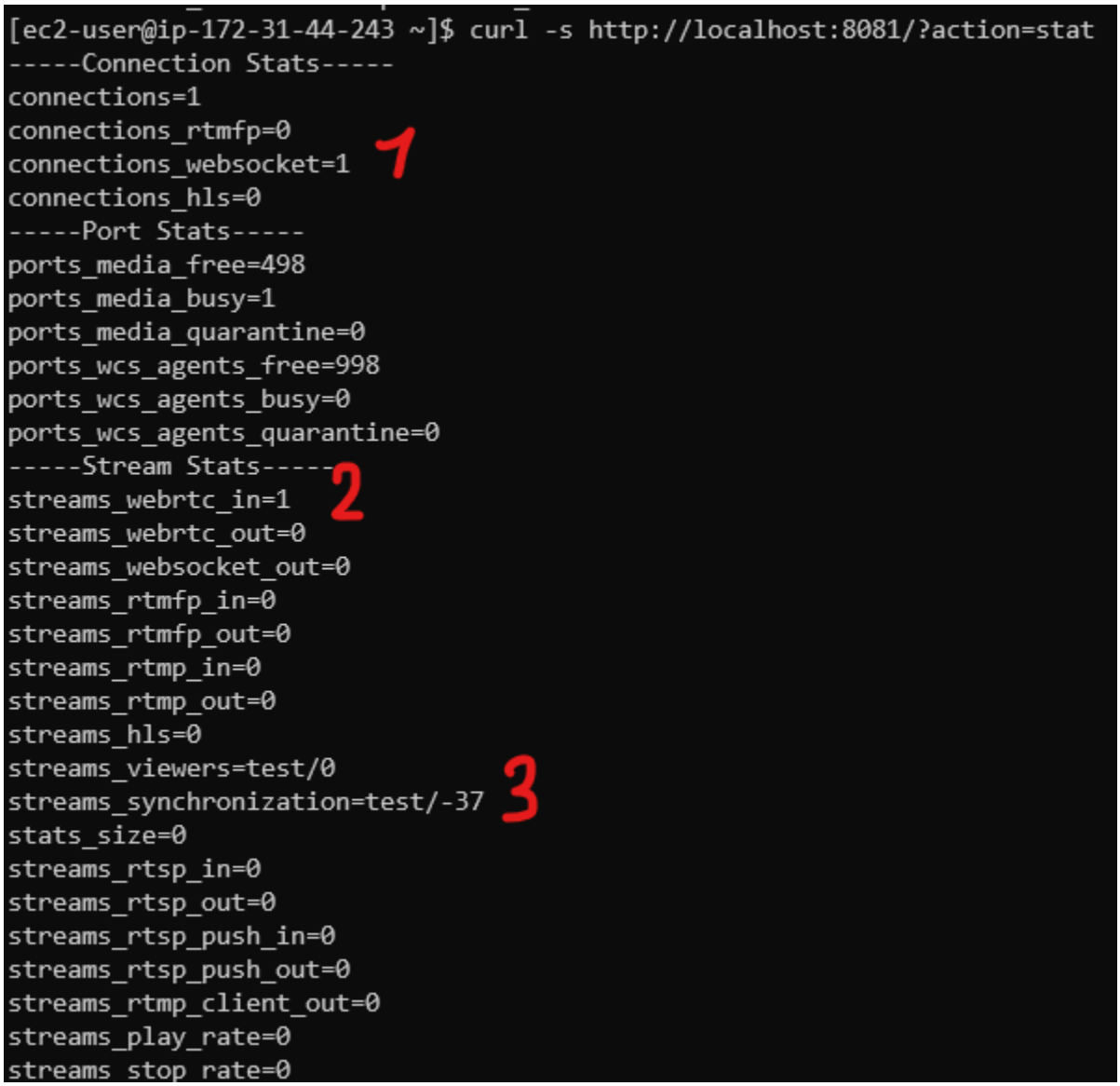
```
sudo systemctl restart haproxy
```

Тестирование

1. Открываем пример Two Way Streaming, указываем порт 443 в поле ввода Websocket URL и публикуем поток



2. Проверяем статистику на первом WCS сервере

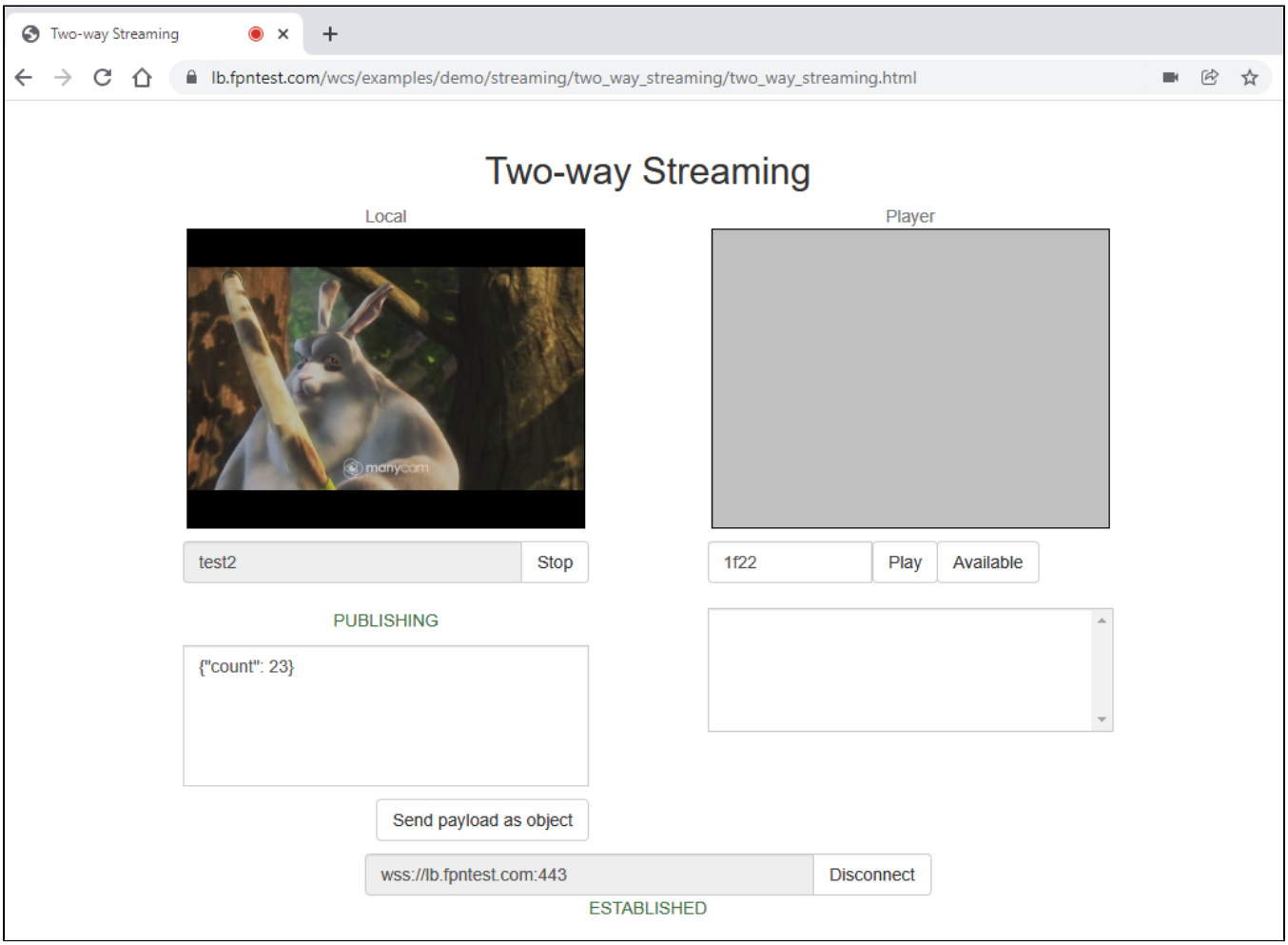


В статистике отображается одно Websocket соединение (1), один входящий поток (2), имя потока `test` (3)

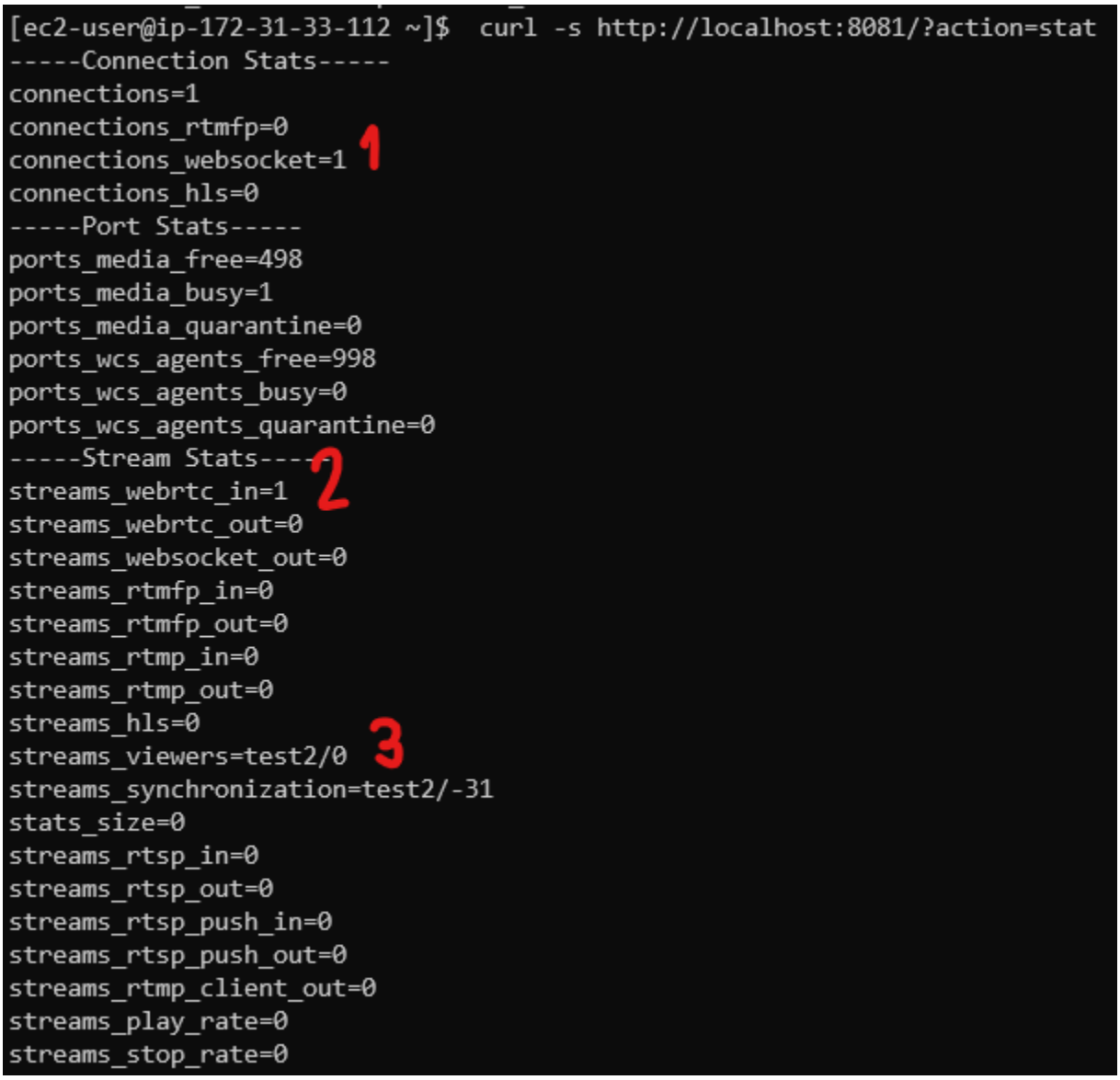
3. Проверяем идентификатор сессии

В идентификаторе сессии указывается IP адрес и порт клиента

4. Открываем пример Two Way Streaming в другом окне, указываем порт 443 в поле ввода Websocket URL и публикуем поток с другим именем



5. Проверяем статистику на втором WCS сервере



В статистике отображается одно Websocket соединение (1), один входящий поток (2), имя потока `test2` (3)