Dynamic Streaming via nginx

Student:	Lukas Abegg
Project:	Multimedia Project 2
Professor:	Sven Spielvogel
Semester:	WiSe 15 / 16
University:	Beuth Hochschule für Technik Luxemburger Str. 10, 13353 Berlin

Table of Contents

Foreword

This project is a term paper of Medienprojekt 2 at Beuth Hochschule für Technik. In this paper will be explained, how this solution can be achieved. How the solution could be implemented and how this solutions works.

It includes the installation guide of the implemented solution, including explanations of all used software and hardware resources, the concept behind and the test implementation of this CDN.

What should be achieved by this project

As initial state, there was the requirement to create a high scalable dynamic streaming solution (CDN) for Live-Broadcasts offered by smartphones, cameras and other mobile devices via apps like Flimme.

Which resources are used?

We are using the following resources:

amazon web services™	**Amazon Web Services** Our CDN solution is implemented as Cloud solution based on AWS. The advantage of this solution is to use as much hardware instances as we want at what time we want. AWS offers already Auto Scaling (scalable instance count based on rules), and Load Balancing (fair sharing of offered server instances).
WOWZA™ media systems	**Wowza Media Systems** We are using Wowza Media System as basic CDN for delivery the content. Wowza already offers testing VOD content without any further configurations and is used by many streaming solutions.
NGINX	**NGINX** It is a very small webserver solution for implementing any web solutions. This server implements many web protocolls. For our requirements we are using HTTP(redirecting) and RTMP(streaming) protocol.

Lua	**Lua Scripts** Lua Scriptings offers many HTTP functions to read request body arguments and a driver to handling database queries. NGINX does not offer the requested functionalities by himself. So we are using Lua Scripts to get this functionalities.
mongolab	**Mongolab** Mongolab offers MongoDB’s to testing our CDN. It offers many drivers to different programming languages, also to Lua Scripts. So we can use this driver for reading our stream offering Wowza Server node to a requested stream via a MongoDB.

Installation Guide OpenResty

This an installation guide to install and configure nginx based streaming edges for Wowza streaming solutions.

What is OpenResty?

OpenResty is nginx based bundle of software including a standard nginx core, LuaJIT, many carefully written Lua libraries, lots of 3rd-party nginx modules, and most of their external dependencies.

Web-Link:	https://openresty.org/#Download

What is Lua?

Lua is a small scripting language which allows to handle and manage http-requests on nginx. In our case, we will use Lua to dynamically pull or push streams with **on_play** and **on_publish** from nginx rtmp-module.

What is the nginx rtmp module?

This module is needed to stream videos to rtmp-clients. In our case we use a flash-player to stream Adobe RTMP.

Github-Link:	https://github.com/arut/nginx-rtmp-module

What is the nginx lua-resty-mongol module?

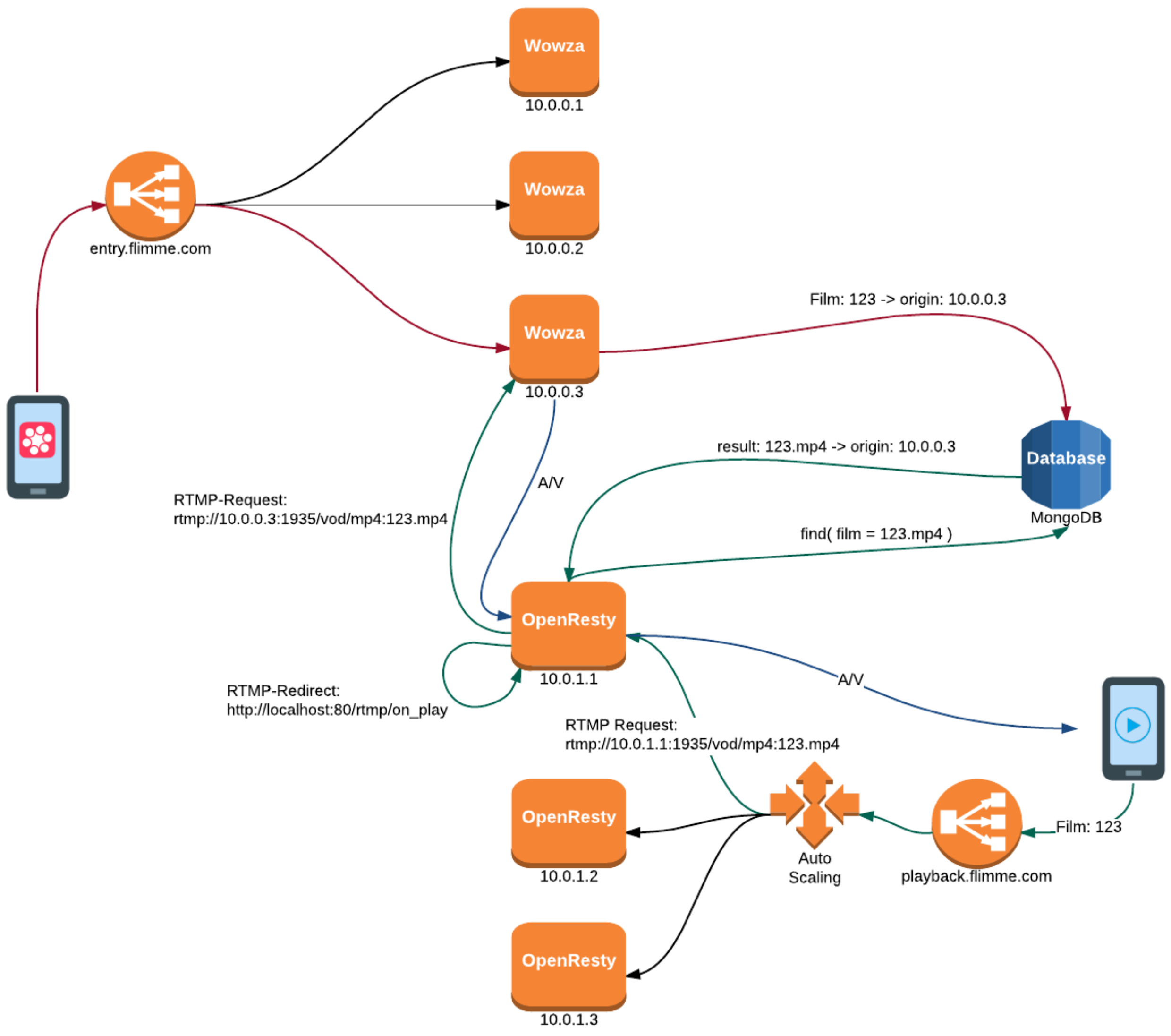
This module is a mongo driver to connect to a MongoDB. We use a MongoDB to save our streaming servers (Wowza's) identity and assigning specific streams to the Wowza-server which offers the stream. This is a security issues to make sure, our nginx edges are not open CDNs(Content Delivery Networks), to stream from wherever the client want, whatever the client want.

Github-Link:	https://github.com/aaashun/lua-resty-mongol

How it should work?

When a client send a rtmp-request to nginx to receive a stream, nginx connects to the MongoDB via this driver and try to find the Wowza server which offers this stream. If nginx find a Wowza server, he takes the IP of the Wowza-Server and redirects to this IP with the rtmp-request he receives from the client. The nginx streams only streams, which are documented in the MongoDB.

Flow Chart of the desired dynamic streaming solution



Installation

Our installation guide is based on a ubuntu 14.04 with ports 80 and 1935 opened for http and rtmp.

We are starting with download all needed packages and unpack them:
(ngx-openrest version: 1.9.7.1)

```
#preparing system
sudo apt-get update
sudo apt-get upgrade

sudo apt-get install build-essential libpcre3 libpcre3-dev libssl-dev
sudo apt-get install libreadline-dev libncurses5-dev perl make build-essential
sudo apt-get install unzip

wget https://openresty.org/download/ngx_openresty-1.9.7.1.tar.gz
wget https://github.com/arut/nginx-rtmp-module/archive/master.zip

tar xvf ngx_openresty-1.9.7.1.tar.gz
unzip master.zip

rm -f master.zip
rm -f ngx_openresty-1.9.7.1.tar.gz

mv nginx-rtmp-module-master ngx_openresty-1.9.7.1/bundle/

cd ngx_openresty-1.9.7.1

./configure --with-luajit \
            --with-pcre-jit \
            --with-ipv6 \
            --without-http_redis2_module \
            --with-http_iconv_module \
            --add-module=bundle/nginx-rtmp-module-master \
            -j2

sudo make
sudo make install

----- Mongo DB Driver ------
wget https://github.com/aaashun/lua-resty-mongol/archive/master.zip
unzip master.zip

rm -f master.zip

cd lua-resty-mongol-master

sudo make install
```

If that worked successful, then you have installed OpenResty with all needed packages.

Now we can start and stop nginx via command line:

start nginx	sudo /usr/local/openresty/nginx/sbin/nginx -c /usr/local/openresty/nginx/conf/nginx.conf
stop nginx	sudo /usr/local/openresty/nginx/sbin/nginx -s stop
reload nginx	sudo /usr/local/openresty/nginx/sbin/nginx -s reload

Create init script for nginx service

Now we need to create some configs to do have a startscript for nginx instead of a long command line.

```
#copy/download/curl/wget the init script
sudo nano /etc/init.d/nginx
```

Copy following script into nginx file:

```
#!/bin/sh
#
# chkconfig: 2345 55 25
# Description: Nginx init.d script, put in /etc/init.d, chmod +x /etc/init.d/nginx
#              For Debian, run: update-rc.d -f nginx defaults
#              For CentOS, run: chkconfig --add nginx
#
### BEGIN INIT INFO
# Provides:          nginx
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: nginx init.d script
# Description:       OpenResty (aka. ngx_openresty) is a full-fledged web application server by bundling the
standard Nginx core, lots of 3rd-party Nginx modules, as well as most of their external dependencies.
### END INIT INFO
#

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DESC="Nginx Daemon"
NAME=nginx
PREFIX=/usr/local/openresty/nginx
DAEMON=$PREFIX/sbin/$NAME
CONF=$PREFIX/conf/$NAME.conf
PID=$PREFIX/logs/$NAME.pid
SCRIPT=/etc/init.d/$NAME

if [ ! -x "$DAEMON" ] || [ ! -f "$CONF" ]; then
    echo -e "\033[33m $DAEMON has no permission to run. \033[0m"
    echo -e "\033[33m Or $CONF doesn't exist. \033[0m"
    sleep 1
    exit 1
fi

do_start() {
    if [ -f $PID ]; then
        echo -e "\033[33m $PID already exists. \033[0m"
        echo -e "\033[33m $DESC is already running or crashed. \033[0m"
        echo -e "\033[32m $DESC Reopening $CONF ... \033[0m"
        $DAEMON -s reopen -c $CONF
        sleep 1
        echo -e "\033[36m $DESC reopened. \033[0m"
    else
        echo -e "\033[32m $DESC Starting $CONF ... \033[0m"
        $DAEMON -c $CONF
        sleep 1
        echo -e "\033[36m $DESC started. \033[0m"
    fi
}

do_stop() {
    if [ ! -f $PID ]; then
        echo -e "\033[33m $PID doesn't exist. \033[0m"
        echo -e "\033[33m $DESC isn't running. \033[0m"
    else
        echo -e "\033[32m $DESC Stopping $CONF ... \033[0m"
        $DAEMON -s stop -c $CONF
        sleep 1
        echo -e "\033[36m $DESC stopped. \033[0m"
    fi
}

do_reload() {
    if [ ! -f $PID ]; then
        echo -e "\033[33m $PID doesn't exist. \033[0m"
        echo -e "\033[33m $DESC isn't running. \033[0m"
        echo -e "\033[32m $DESC Starting $CONF ... \033[0m"
        $DAEMON -c $CONF
        sleep 1
        echo -e "\033[36m $DESC started. \033[0m"
    else
        echo -e "\033[32m $DESC Reloading $CONF ... \033[0m"
        $DAEMON -s reload -c $CONF
        sleep 1
        echo -e "\033[36m $DESC reloaded. \033[0m"
    fi
}

do_quit() {
    if [ ! -f $PID ]; then
        echo -e "\033[33m $PID doesn't exist. \033[0m"
        echo -e "\033[33m $DESC isn't running. \033[0m"
    else
        echo -e "\033[32m $DESC Quitting $CONF ... \033[0m"
        $DAEMON -s quit -c $CONF
        sleep 1
        echo -e "\033[36m $DESC quitted. \033[0m"
    fi
}

do_test() {
    echo -e "\033[32m $DESC Testing $CONF ... \033[0m"
    $DAEMON -t -c $CONF
}

do_info() {
    $DAEMON -V
}

case "$1" in
 start)
 do_start
 ;;
 stop)
 do_stop
 ;;
 reload)
 do_reload
 ;;
 restart)
 do_stop
 do_start
 ;;
 quit)
 do_quit
 ;;
 test)
 do_test
 ;;
 info)
 do_info
 ;;
 *)
 echo "Usage: $SCRIPT {start|stop|reload|restart|quit|test|info}"
 exit 2
 ;;
esac

exit 0
```

Copy that script to the **/etc/init.d/** directory and make it executable:

```
sudo chmod +x /etc/init.d/nginx
```

Once you are done editing the file, run the following command to set it up:

```
sudo update-rc.d nginx defaults
```

Now you can simple start and stop the service:

- starting Nginx: sudo service nginx start
- stopping Nginx: sudo service nginx stop
- restart Nginx: sudo service nginx restart
- run a syntax test on the configuration file: sudo service nginx test

Here are some sample outputs:

```
ubuntu@ip-172-31-28-79:/usr/local$ sudo service nginx stop
-e  /usr/local/openresty/nginx/logs/nginx.pid doesn't exist.
-e  Nginx Daemon isn't running.

ubuntu@ip-172-31-28-79:/usr/local$ sudo service nginx start
-e  Nginx Daemon Starting /usr/local/openresty/nginx/conf/nginx.conf ...
-e  Nginx Daemon started.
```

Configure rtmp-module and dynamic pull for streaming

After installation und configure of nginx you are ready to implement dynamic pull streaming via rtmp-module.

For that you have to configure the nginx. Nginx has got an own configuration file placed in:

```
/usr/local/nginx/conf/nginx.conf
```

Edit nginx.conf:

```
sudo nano /usr/local/openresty/nginx/conf/nginx.conf
```

Basically you have to setup standard host configuration like this:

```
# user  nobody;
worker_processes  1;

# activate log file in /usr/local/openresty/nginx/logs/error.log
error_log  logs/error.log;

# nginx process id
pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

# setup http-port 80
http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;
        access_log  logs/host.access.log;

        location / {
            root   html;
            index  index.html index.htm;
        }

        #error_page  404              /404.html;
        # redirect server error pages to the static page /50x.html
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```

After start of nginx you can test if your server is running via curl:

```
curl localhost:80
```

You should receive the standard welcome page like that:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Now you are ready to configure your rtmp-module.

First you have to add an rtmp-module configuration:

```
#RTMP
rtmp {
        server {
                listen 1935;

                # Dynamic pull via http lua rewrite from “/rtmp/on_play”
                application vod {
                            live on;
                            on_play http://localhost:80/rtmp/on_play;
                }
          }
}
```

In this rtmp-module you initiate the application **vod** which makes **on_play** a dynamic pull from the local http method **rtmp/on_play**. By the dynamic pull you send all http-post arguments within your body.

In addition you have to add a http-function **rtmp/on_play**:

```
location = /rtmp/on_play {

    rewrite_by_lua '
            ngx.req.read_body()
            local args = ngx.req.get_post_args()

            local origin
            local app
            local name

                for key, val in pairs(args) do
                 if type(val) == "table" then
                    ngx.log( ngx.ALERT, key, ": " , table.concat(val, ", "))
                 else
                    ngx.log( ngx.ALERT, key, ": " , val)
                 end

                 if     key == "origin"      then origin = val
                 elseif key == "app"         then app = val
                      elseif key == "name"          then name = val
                 end
            end

            ngx.log( ngx.ALERT, "check this: origin =  ", origin)
            ngx.log( ngx.ALERT, "check this: app =  ", app)
            ngx.log( ngx.ALERT, "check this: name =  ", name)

            local uri = "rtmp://" .. origin .. "/" .. app .. "/" .. name .."?"

            ngx.log( ngx.ALERT, "check this: uri =  ", uri)
            return ngx.redirect(uri);
    ';
}
```

Here is a description of the coding:

Code	**Description**

rewrite_by_lua '
        ...
';
	start lua scripting

ngx.req.read_body() local args = ngx.req.get_post_args()	before you can read post args you have to read the request body
local origin local app local name	define local vars for arguments
for key, val in pairs(args) do ... end	loop over all keys / values from post request

if type(val) == "table" then
   ngx.say(key, ": " , table.concat(val, ", "))
else
   ngx.say(key, ": " , val)
end
	log all arguments

if   key == "origin"
   then origin = val
elseif key == "app"
   then app = val
elseif key == "name"
   then name = val
end
	set all arguments in vars

local uri = "rtmp://" .. origin .. "/" .. app .. "/" .. name .. "?"	concatenate uri for rtmp-stream: rtmp://origin/app/name
return ngx.redirect(uri);	rewrite pull by lua redirect of new uri

Configure lua-resty-mongol for Mongo-DB Connection

At the moment our nginx is like an open CDN-Edge. You can stream whatever you want from wherever you want. This is very risky to getting hacked. For that we want to read the origin stream server (Wowza) from a Mongo-DB. For that we did install lua-resty-mongol. This module includes a mongo-driver and a language package for lua to sending and receiving data from a Mongo-DB.

Authentication based on MongoDB-Version

- SCRAM-SHA-1
  - Is the default mechanism for MongoDB versions beginning with the 3.0 series. Its a salt cryptology based authentication mechanism.
- MONGODB-CR
  - Is a challenge-response mechanism that authenticates users through passwords.

Configuration

To use this package, we have to configure again the nginx.conf.

Edit nginx.conf:

```
sudo nano /usr/local/openresty/nginx/conf/nginx.conf
```

To clearify how MongoDB connection works, here an example implementation which renders a html page with the query result.

Here are the details for that constellation:

MongoDB	Host = ds037005.mongolab.com (54.170.75.122)
	Port = 37005

Authentication	Mechanism = SCRAM-SHA-1

	User = wowza_admin
	Password = Wowza_Admin15

Database	mongo_wowza_streams
Collection	streams

Document-Structure	`{`
	`"_id": {`
	`"$oid": "56b37afae4b0102fef244213"`
	`},`
	`"film": "sample.mp4",`
	`"origin": "52.29.2.49"`
	`}`

And here is the implementation code:

```
location = /mongo {
    default_type text/html;

    content_by_lua '

        local mongo = require "resty.mongol"
        conn = mongo:new()
        conn:set_timeout(1000)
        ok, err = conn:connect("54.170.75.122", 37005)

        if not ok then
                ngx.say("connect failed: "..err)
        else
                ngx.say("connected: "..ok)
                local db = conn:new_db_handle("mongo_wowza_streams")
                ok, err = db:auth_scram_sha1("wowza_admin","Wowza_Admin15")

                if not ok then
                ngx.say("<br>authentication failed: "..err)
                else
                ngx.say("<br>logged in: "..ok)

                col = db:get_col("streams")
                r = col:find_one({film="sample.mp4"})

                ngx.say("<br><b>result</b>")
                ngx.say("<br>id: "..r["_id"]:tostring())
                ngx.say("<br>film: "..r["film"])
                ngx.say("<br>origin: "..r["origin"])
                end

        end
    ';
}
```

By running this request:

```
http://*IP-NGINX*/mongo
```

You should get the following result page:

```
connected: 1
logged in: 1
result
id: 56b37afae4b0102fef244213
film: sample.mp4
origin: 52.29.2.49
```

Now we know the module works and we get the right answer. So we can start including that function into our code to **/rtmp/on_play**:

```
location = /rtmp/on_play {

rewrite_by_lua '
            ngx.req.read_body()
            local args = ngx.req.get_post_args()

            local origin
            local app
            local name

            for key, val in pairs(args) do
            if          key == "app"       then app = val
            elseif key == "name"        then name = val
            end
            end

            local mongo = require "resty.mongol"
            local result

            local film = string.gsub(name, "mp4:", "")
            ngx.log( ngx.ALERT, "search stream for film: "..film)

            conn = mongo:new()
            conn:set_timeout(1000)
            ok, err = conn:connect("54.170.75.122", 37005)

            if not ok then
             ngx.log( ngx.ALERT, "connect to mongodb failed: "..err)
            else
             ngx.log( ngx.ALERT, "connected to mongodb: "..ok)

             local db = conn:new_db_handle("mongo_wowza_streams")
             ok, err = db:auth_scram_sha1("wowza_admin","Wowza_Admin15")

             if not ok then
            ngx.log( ngx.ALERT, "authentication database failed: "..err)
             else
            ngx.log( ngx.ALERT, "logged in database succeded: "..ok)

            col = db:get_col("streams")
            result = col:find_one({film=film})

            ngx.log( ngx.ALERT, "mongodb result (id): "..result["_id"]:tostring())
            ngx.log( ngx.ALERT, "mongodb result (film): "..result["film"])
            ngx.log( ngx.ALERT, "mongodb result (origin): "..result["origin"])
            origin = result["origin"]

             end

            end

            ngx.log( ngx.ALERT, "redirect with this: origin =  ", origin)
            ngx.log( ngx.ALERT, "redirect with this: app =  ", app)
            ngx.log( ngx.ALERT, "redirect with this: name =  ", name)

            local uri = "rtmp://" .. origin .. "/" .. app .. "/" .. name .. "?"

            ngx.log( ngx.ALERT, "redirect with this: uri =  ", uri)
            return ngx.redirect(uri);
';
}
```

Here is a description of the coding:

Code	**Description**
`local mongo = require "resty.mongol"`	Initialise mongo driver to use in coding
`local film = string.gsub(name, "mp4:", "")`	Read stream out of argument name

`conn = mongo:new()`
`conn:set_timeout(1000)`
	Open new database connection and set timeout time for answering

`ok, err = conn:connect("54.170.75.122", 37005)`	Connect to mongolab
`conn:new_db_handle("mongo_wowza_streams")`	Open handler for database on mongolab
`db:auth_scram_sha1(<user>,<password>)`	Authentication on database
`col = db:get_col("streams")`	Use collection “streams” for queries
`r = col:find_one({film="sample.mp4"})`	Get document with attribute film equals the requested value
`r["_id"]:tostring())`	Get stringified id of document “r”
`r["film"]`	Get key “film” of document “r”

Example Configuration

This is an example configuration I did. In this example you can see how it works for:

Function()	**Description**
rtmp > application vod {}	rtmp dynamic pull via http lua redirect
rtmp > application vod2 {}	rtmp static rewrite
http > location /rewrite {}	http rewrite with post arguments
http > location / {}	http standard response on port 80
http > location = /echo {}	http echo from post arguments
http > location = /mongo {}	http lua mongodb connection with query result rendering
http > location = /rtmp/on_play {}	http lua redirect with mongodb driver and post arguments

Implementation:

```
worker_processes  1;

error_log  logs/error.log;
pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

#HTTP
http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

        rewrite_log on;

              location = /rtmp/on_play {

              rewrite_by_lua '
                            ngx.req.read_body()
                            local args = ngx.req.get_post_args()

                            local origin
                            local app
                            local name

                            for key, val in pairs(args) do
                            if          key == "app"      then app = val
                            elseif key == "name"        then name = val
                            end
                            end

                            local mongo = require "resty.mongol"
                            local result

                            local film = string.gsub(name, "mp4:", "")
                            ngx.log( ngx.ALERT, "search stream for film: "..film)

                            conn = mongo:new()
                            conn:set_timeout(1000)
                            ok, err = conn:connect("54.170.75.122", 37005)

                            if not ok then
                             ngx.log( ngx.ALERT, "connect to mongodb failed: "..err)
                            else
                             ngx.log( ngx.ALERT, "connected to mongodb: "..ok)

                             local db = conn:new_db_handle("mongo_wowza_streams")
                             ok, err = db:auth_scram_sha1("wowza_admin","Wowza_Admin15")

                             if not ok then
                            ngx.log( ngx.ALERT, "authentication database failed: "..err)
                             else
                            ngx.log( ngx.ALERT, "logged in database succeded: "..ok)

                            col = db:get_col("streams")
                            result = col:find_one({film=film})

                            ngx.log( ngx.ALERT, "mongodb result (id): "..result["_id"]:tostring())
                            ngx.log( ngx.ALERT, "mongodb result (film): "..result["film"])
                            ngx.log( ngx.ALERT, "mongodb result (origin): "..result["origin"])
                            origin = result["origin"]

                             end

                            end

                            ngx.log( ngx.ALERT, "redirect with this: origin =  ", origin)
                            ngx.log( ngx.ALERT, "redirect with this: app =  ", app)
                            ngx.log( ngx.ALERT, "redirect with this: name =  ", name)

                            local uri = "rtmp://" .. origin .. "/" .. app .. "/" .. name .. "?"

                            ngx.log( ngx.ALERT, "redirect with this: uri =  ", uri)
                            return ngx.redirect(uri);
              ';
              }

              location = /mongo {
              default_type text/html;

              content_by_lua '

              local mongo = require "resty.mongol"
              conn = mongo:new()
              conn:set_timeout(1000)
              ok, err = conn:connect("54.170.75.122", 37005)

              if not ok then
                            ngx.say("connect failed: "..err)
              else
```

```
                        ngx.say("connected: "..ok)
                        local db = conn:new_db_handle("mongo_wowza_streams")
                        ok, err = db:auth_scram_sha1("wowza_admin","Wowza_Admin15")

                        if not ok then
                        ngx.say("<br>authentication failed: "..err)
                        else
                        ngx.say("<br>logged in: "..ok)

                        col = db:get_col("streams")
                        r = col:find_one({film="sample.mp4"})

                        ngx.say("<br><b>result</b>")
                        ngx.say("<br>id: "..r["_id"]:tostring())
                        ngx.say("<br>film: "..r["film"])
                        ngx.say("<br>origin: "..r["origin"])
                        end

            end
            ';
            }

        location /rewrite {
            rewrite ^.*$ rtmp://52.28.135.233/vod? permanent;
        }

        location = /echo {
            set_unescape_uri $origin $arg_origin;
            echo "Hello, $origin!";
        }

        location / {
            default_type text/html;
            content_by_lua '
                ngx.say("<p>hello, world</p>")
            ';
        }
    }
}

#RTMP
rtmp {
    server {
        listen 1935;

        # Dynamic pull via http lua rewrite
        application vod {
            live on;
            on_play http://localhost:80/rtmp/on_play;
        }

        # Nginx static rewrite
        application vod2 {
            live on;
            on_play http://localhost:80/rewrite;
        }
    }
}
```

Prepare Streaming Network for Testing

What do we need?

- **Hardware**
  - Cloud Solution -> Amazon Web Services (AWS)
  - Database -> mongolab.com

- **Software**
  - Streaming Server (Node)
    - Basis Cloud Images -> Ubuntu 14.04
    - Streaming Server -> Wowza Streaming Server

  - Streaming Edge Server
    - Basis Cloud Images -> Ubuntu 14.04
    - Streaming Server -> Nginx based OpenResty
    - Additional Packages
      - RTMP-Streaming -> nginx-rtmp-module
      - MongoDB-Driver -> lua-resty-mongol

- **AWS Configuration**
  - Security Group -> Permission Rules for In-/Outbound of servers
  - Load Balancer -> Balancing between Edge Instances
  - Auto Scaling Group -> Managing number of server instances
  - Images(AMI) -> Complete configured images of our servers

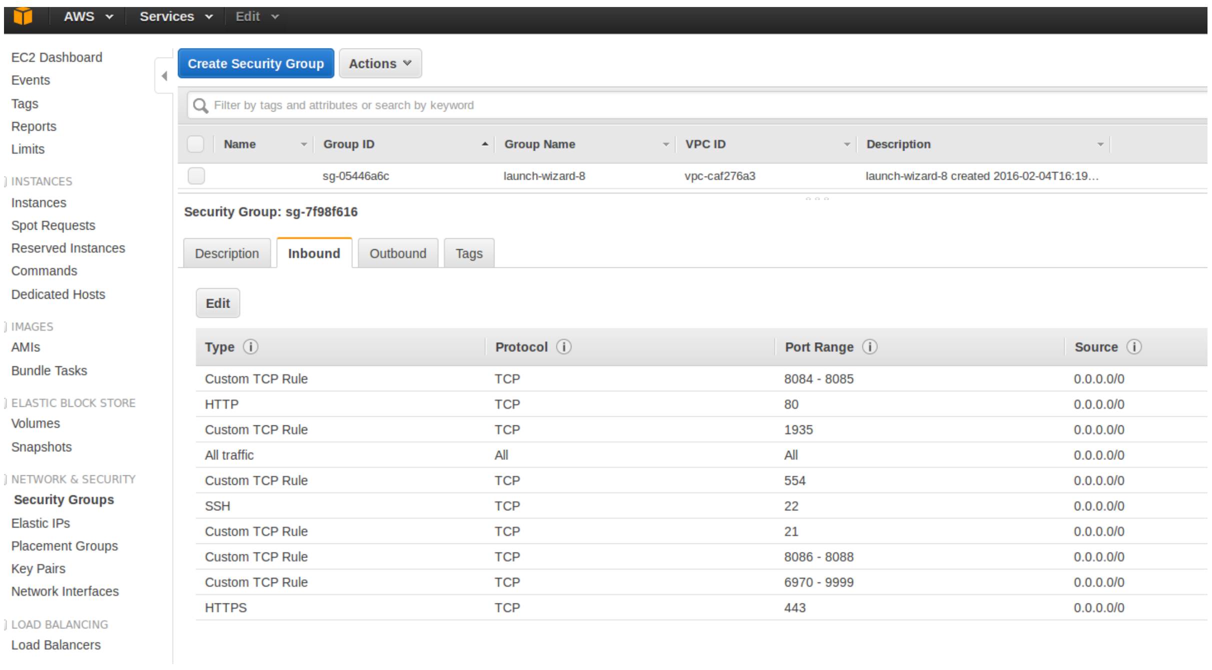
- **Mongolab**
  - Stream Database -> Collection of streams and their streaming node

Creating Security Group

First of all we need a Security Group to open needed ports for streaming. For that we create a new Security Group in AWS > EC2.

Most important is to open the port:

● 80/443	HTTP (Redirects)
● 1935	RTMP (Streaming)
● 22	SSH (Connection to ssh-client)
● 37005	mongolab.com (Database)



Name	Group ID	Group Name	VPC ID	Description
	sg-05446a6c	launch-wizard-8	vpc-caf276a3	launch-wizard-8 created 2016-02-04T16:19…

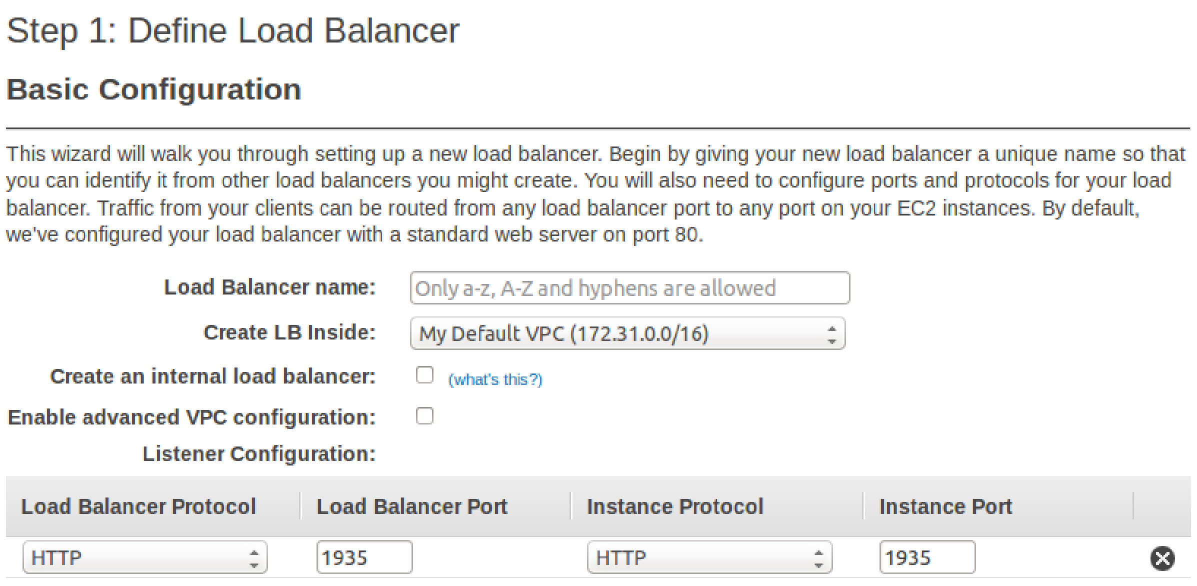
Security Group: sg-7f98f616

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	8084 - 8085	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0
Custom TCP Rule	TCP	1935	0.0.0.0/0
All traffic	All	All	0.0.0.0/0
Custom TCP Rule	TCP	554	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	21	0.0.0.0/0
Custom TCP Rule	TCP	8086 - 8088	0.0.0.0/0
Custom TCP Rule	TCP	6970 - 9999	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

Creating Load Balancer

To balance the streaming clients between the running instances of nginx edges, we can use the AWS Load Balancer. A load balancer offers us the possibility to balance the load and having well loaded edge instances.

Because we are rtmp streaming, we have to balance the load on port 1935 (RTMP communication port).



Step 1: Define Load Balancer

Basic Configuration

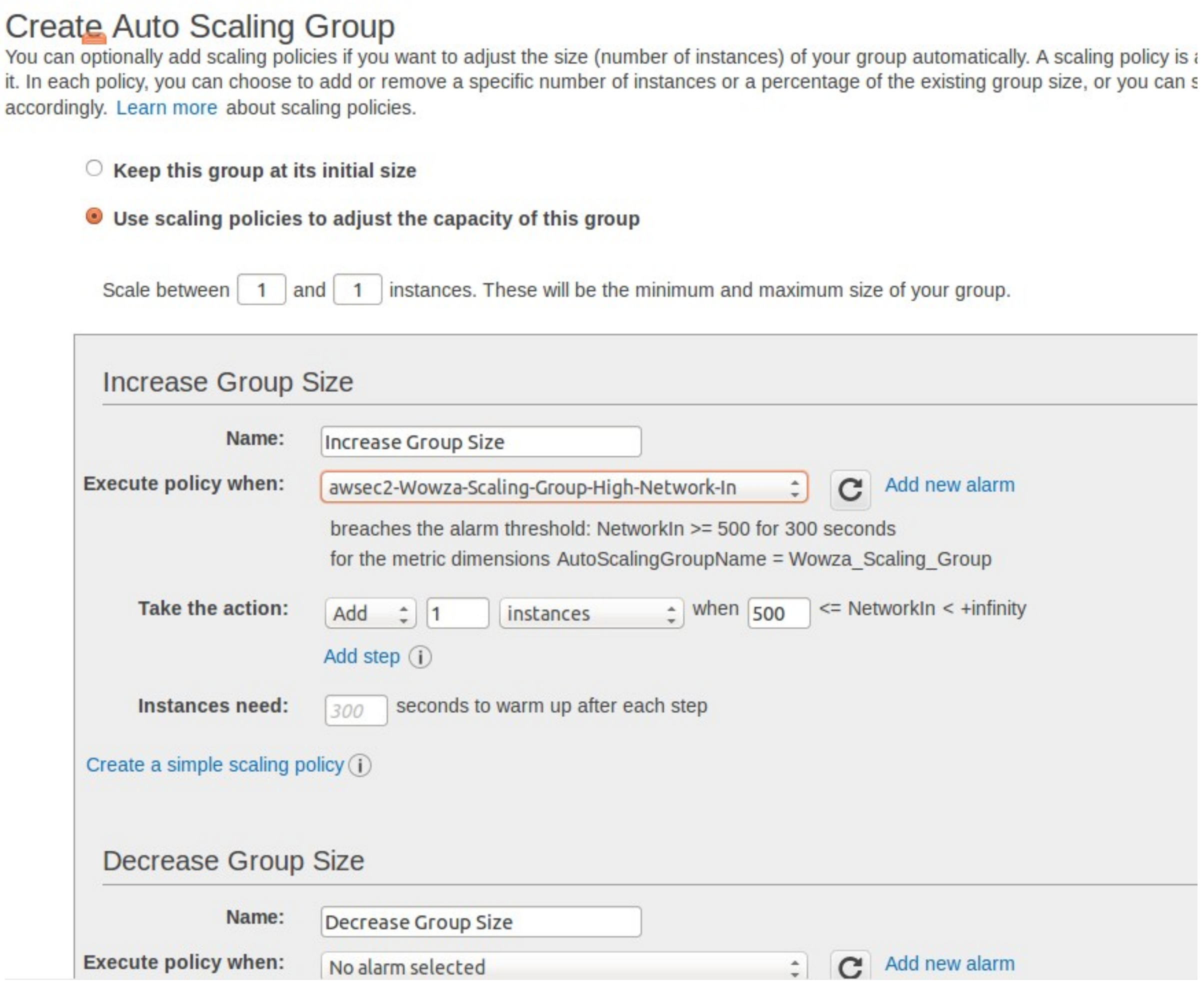
This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name:	Only a-z, A-Z and hyphens are allowed
Create LB Inside:	My Default VPC (172.31.0.0/16)
Create an internal load balancer:	☐ (what's this?)
Enable advanced VPC configuration:	☐
Listener Configuration:	

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	
HTTP	1935	HTTP	1935	⊗

Creating Auto Scaling Group

This tool by AWS offers us a great opportunity to define rules, in which case we want to increase the number of nginx edges and wowza nodes. That means, we can dynamic manage, how many instance of edges and nodes we are running depending on as example CPU load or better in our case network load.



Create Auto Scaling Group

You can optionally add scaling policies if you want to adjust the size (number of instances) of your group automatically. A scaling policy is a
it. In each policy, you can choose to add or remove a specific number of instances or a percentage of the existing group size, or you can s
accordingly. Learn more about scaling policies.

○ Keep this group at its initial size

● Use scaling policies to adjust the capacity of this group

Scale between 1 and 1 instances. These will be the minimum and maximum size of your group.

Increase Group Size

Name: Increase Group Size

Execute policy when: awsec2-Wowza-Scaling-Group-High-Network-In

breaches the alarm threshold: NetworkIn >= 500 for 300 seconds
for the metric dimensions AutoScalingGroupName = Wowza_Scaling_Group

Take the action: Add 1 instances when 500 <= NetworkIn < +infinity

Instances need: 300 seconds to warm up after each step

Decrease Group Size

Name: Decrease Group Size

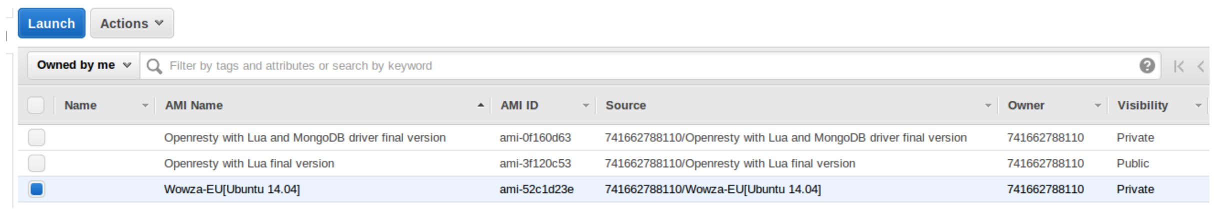
Execute policy when: No alarm selected

AMIs

With AMIs we can prepare full configured images of our node and edge servers. That means, if we want to run our server, we just have to launch this AMIs.

I made 3 versions:

AMI	**AMI ID**	**Description**
Openresty with Lua and MongoDB driver final version	ami-0f160d63	Full configured edge with open resty installation including all modules and database connection
Openresty with Lua final version	ami-3f120c53	Configured edge with open resty installation including rtmp module and redirection via POST-arguments
Wowza-EU[Ubuntu 14.04]	ami-52c1d23e	Wowza example configuration to test nginx edges



Name	AMI Name	AMI ID	Source	Owner	Visibility
	Openresty with Lua and MongoDB driver final version	ami-0f160d63	741662788110/Openresty with Lua and MongoDB driver final version	741662788110	Private
	Openresty with Lua final version	ami-3f120c53	741662788110/Openresty with Lua final version	741662788110	Public
	Wowza-EU[Ubuntu 14.04]	ami-52c1d23e	741662788110/Wowza-EU[Ubuntu 14.04]	741662788110	Private

Mongolab

Mongolab offers mongo databases to testing our streaming solution. For our test case we need the following constellation:

● MongoDB URL	->	ds037005.mongolab.com
● Access Port	->	37005
● Database	->	mongo_wowza_streams
● Collection	->	streams
● Documents	->	an example record of our stream

Example Record:

```
{
        "_id": {
        "$oid": "56b37afae4b0102fef244213"
        },
        "film": "sample.mp4",
        "origin": "52.29.234.12"
}
```

How to connect to the database

To connect using the mongo shell:

```
mongo ds037005.mongolab.com:37005/mongo_wowza_streams -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI:

```
mongodb://<dbuser>:<dbpassword>@ds037005.mongolab.com:37005/mongo_wowza_streams
```

Example Stream-Constellation

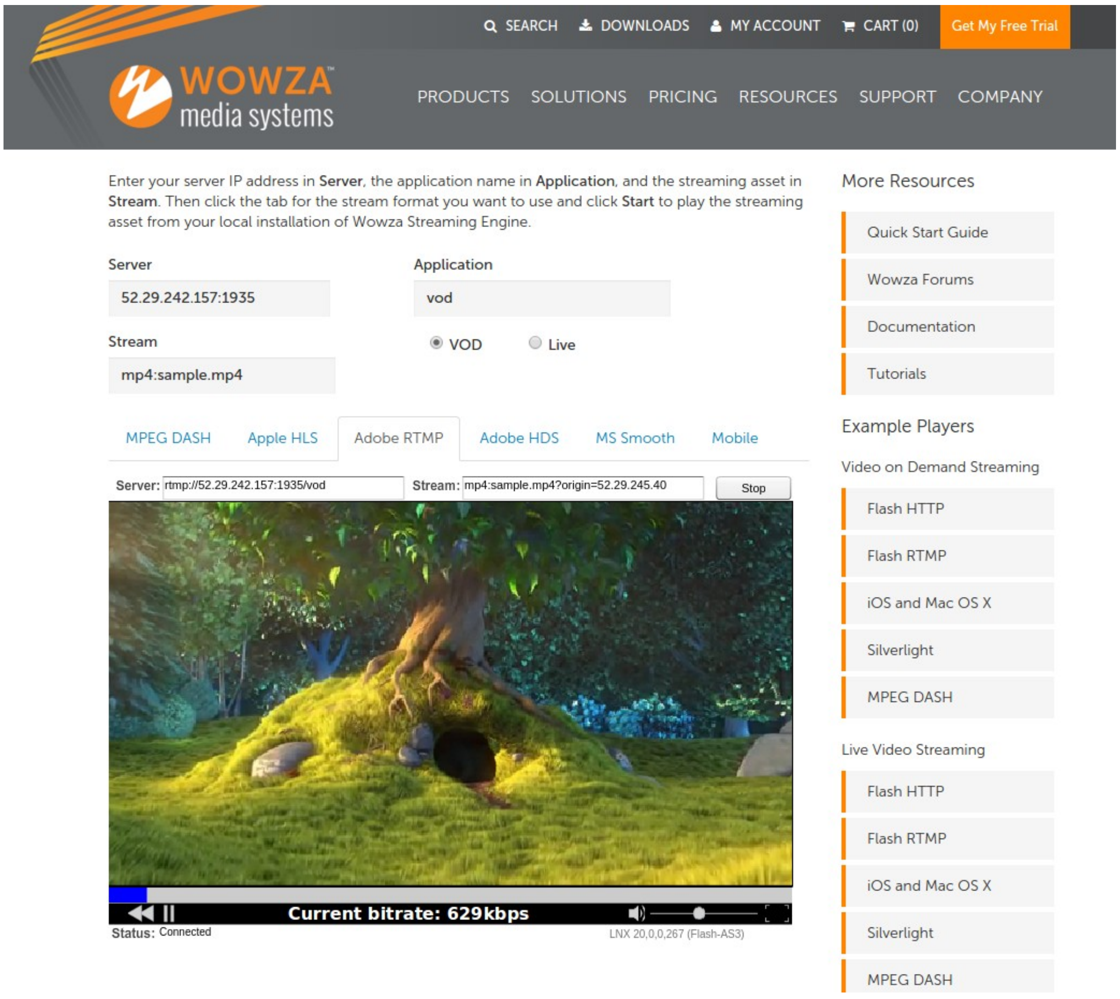
This is the constellation we use for this example:

nginx edge public IP	*52.29.242.157*
wowza node public IP:	*52.29.245.40*
example stream	*mp4:sample.mp4*
aplication of stream	*vod*
testplayer	*https://www.wowza.com/testplayers*

Test-Request

rtmp://52.29.242.157:1935/vod/mp4:sample.mp4

Result



Enter your server IP address in **Server**, the application name in **Application**, and the streaming asset in **Stream**. Then click the tab for the stream format you want to use and click **Start** to play the streaming asset from your local installation of Wowza Streaming Engine.

Server: 52.29.242.157:1935
Application: vod
Stream: mp4:sample.mp4
[✓ VOD] Live

Server: rtmp://52.29.242.157:1935/vod
Stream: mp4:sample.mp4?origin=52.29.245.40

Current bitrate: 629kbps

Status: Connected
LNX 20,0,0,267 (Flash-AS3)