Nimble Advertizer technology description

Nimble Advertizer is a server-side advertisement insertion technology based on Nimble Streamer media server. This document describes technical details of its setup and usage.

1. Basic workflow

Ads insertion technology workflow works as follows:

1. Nimble Streamer media server processes incoming streams to get content. See section 2 for prerequisites.
2. Nimble Advertizer calls your handler web application (see section 3), handler URL is defined in Nimble Streamer config file (see section 4).
3. Nimble Advertizer gets response from handler with ads scenarios (see section 5).
4. Advertizer gets files with advertisements to process them via Nimble Streamer according to ads scenarios logic defined by handler response.
5. Nimble inserts the ads into original media and packages it into outgoing stream.
6. End user connects to Nimble and gets media stream containing original content mixed with advertisements.
7. Currently HLS (both video+audio and audio-only modes), SLDP, RTMP and Icecast are supported as output protocols while all protocols supported by Nimble Streamer can be used as input.

2. Prerequisites

Before following the description below please make sure you comply with the following prerequisites:

1. You have WMSPanel premium account which has active subscription.
2. Your WMSPanel login has admin privileges.
3. You have Nimble Streamer installed and running.
4. You subscribed for Nimble Advertizer, had active license and registered it (see instruction later in this section).
5. Advertising files are packaged into MP4 container. You may use ffmpeg to re-package files into MP4.
6. Advertising video is encoded to H.264.

7. Advertising audio has the same codec and sample rate as your input stream. E.g. if your stream is AAC 44.1KHz, your ads will be packaged into MP4 file with AAC audio with 44.1KHz sample rate.
8. Advertizing files must be accessible for download from Nimble instance via HTTP/HTTPS URL.
9. Input streams are set up and their content is available.
10. For SLDP output: the output stream is set up and is publicly available. Use proper instruction from this page to perform the setup: https://wmspanel.com/nimble/sldp
11. For RTMP output: the output stream is set up and is publicly available. Use proper instruction from this page to perform the setup: https://wmspanel.com/nimble/rtmp
12. For HLS output: the output stream is set up and is publicly available. Use proper instruction from this page to perform the setup: https://wmspanel.com/nimble/hls . Notice that Nimble Streamer application setting must have “HLS” output enabled; “HLS MPEGTS” or “HLS fMP4” are not yet supported.
13. For Icecast ads: input stream has either AAC or MP3 encoded audio. The output will have the same codec.
14. For Icecast: the Icecast output stream is set up and is publicly available. Use proper instruction from this page to perform the setup: https://wmspanel.com/nimble/audio_streaming

To obtain and register the Advertizer license follow these steps:

1. Go to Settings menu and open Advertizer licenses tab
2. Click on New SSAI subscription, select number of licenses, click Next and once you see final cost, click on “Pay now” button.
3. Follow the payment procedure and once you complete it, you’ll see your active licenses.
4. Click on the license to see the registration procedure.
5. Log into the server which has Nimble Streamer installed and which you will use as your Nimble Advertizer host. Then use the registration procedure to make it work.

That’s it, you may now proceed with your Advertizer setup.

3. Handler app

Nimble Advertizer business logic for ads insertion is defined by your handler.
Handler is a REST controller which is called by Nimble Advertizer and responds with ads scenarios. It’s called periodically as defined in Nimble config file, see section 4 for details.
Handler app must be available via HTTP/HTTPS protocol and accessible for from Nimble instance.
You may use any language and framework to create a handler application.
Handler response must be a valid JSON text, its grammar is described in section 5.

4. Config file setup

Nimble Advertizer behavior on server side is defined via Nimble Streamer configuration file. Its general parameters and usage are described in this article:
https://blog.wmspanel.com/p/nimble-streamer-configuration.html

The following settings must be used for serve setup

advertising_url=http://<server_addr>/<handler_app> | https://<server_addr>/<handler_app>
Defines the URL of handler app. Example:
advertising_url=http://testradiowebsite.com/ads_handler.php

advertising_sync_interval=<seconds>
Defines how frequent Nimble Advertizer will request the handler for updates of ads scenarios. It's 10 seconds by default. Example:
advertising_sync_interval=3

max_advertising_download_sessions=<number>
Defines how many concurrent sessions Nimble Advertizer will use for downloading the files which contain ads. It's 10 sessions by default. If handler specifies more files that defined in this parameter, the extra files will be queued and downloaded in their turn Example:
max_advertising_download_sessions=20

Once you define settings in config file, please restart Nimble Streamer. For Linux it's done by "sudo service nimble restart" command.

5. Handler response grammar

Handler application returns a JSON text as a response to server sync-up call. This response describes advertising scenarios for Nimble Advertizer to follow.

Here is a formatted example of response which shows its structure and elements.

```
{
  "contents":
  [
    {"id":"11","uri":"http:\/\/192.168.232.129\/SampleAudio_0.4mb_5s.aac.mp4"},
    {"id":"22","uri":"https:\/\/192.168.232.129\/testt.mp4"},
    {"id":"33","uri":"http:\/\/8.8.8.8\/SampleAudio_0.4mb_5s.mp3.mp4"}
  ],
  "rules":
  [
    {
    "id":"1",
    "protocols":["icecast", "sldp", "rtmp", "hls"],
    "users":[],
    "type":"global",
    "time_sync":"stream",
    "time_offset":0,
    "time_interval":600,
    "contents":[{"id":"11","onerror":"stop","wait":"10"},
{"id":"22","onerror":"skip","wait":"2"}]
    }
  ]
}
```

Response has 2 general elements - “**contents**" and “**rules**".

The “**contents**" element describes the ads files to be used in advertising rules. It’s an array of ads, each having “id" for ad’s ID and “uri" for ad location URI.

The ID is a unique identifier of an ad. Once new ID appears in handler response, it’s downloaded and put into Nimble cache for further insertion into streams. If you’d like to update the ad over time, you need to update your ad file and add it to your response with new ID. In this case the old file will be removed from cache and the new one will be uploaded. Removing ID from response means this ad will be played for all clients where it started to play already and then it will be removed from cache.

The URI is any valid web location available via HTTP or HTTPS.

The “**rules**" element describes the ads insertion logic. Each rule describes applicability of certain ads for certain set of streams or users. If several rules can be applied to the same stream, then all those rules will be applied one by one. The application order is defined by the time of ad appearance defined in each rule. If several ads apply for the same time, then they will be queued and played one after another.

Each rule has its “**id**" element to define its ID. If new ID appears in rules, this rules starts working. If it’s removed then the rule is removed too and it stops working. If you change the rule without changing the ID, this rule’s change will not be applied. So if you’d like to modify the rule - just assign new ID to it.

The “**protocols**" element describes which protocols this rule will apply to. At the moment only "icecast", "sldp", "rtmp" and "hls" values are supported. Notice that HLS is supported only for audio-only mode.

The “**users**" element is populated in case you want to insert personalised ads. It’s a list of IDs of users to apply this rule for. It’s defined by pay-per-view framework described on this page: https://wmspanel.com/nimble/pay_per_view . If you don’t want to personalise ads, just keep this list empty in all rules.

The “**type**" element defines what type of entity this rule is applied. Possible values are “global", “app" and "stream".
“global" type means this rule is applied to all streams on current server.
"app" type means this rule will be applied to a certain streaming application. In this case you also need to add "app" element with application name. E.g. if you have streams like live/stream1, live/stream2 etc. then you can add these elements: “type":"app","app":"live".
"stream" type means the rule will be applied to exact stream. It needs “app" and “stream" elements added to describe the stream. E.g. if you want to apply a rule to live/radio1 stream, then you’ll need to use these elements: “type":"stream","app":"live","stream":"radio1".

The “**time_sync**" element defines the reference point for this rule to define time for ads insertion. It defines “**time_offset**" element usage.

If you use “time_sync":"gmt" value then your ad insertion will be defined by exact time point. “time_offset" will define time of insertion in “%Y-%m-%d %T" format, e.g. 2018-01-01 10:00:00.

If you use “time_sync":"stream" value then you will define start time for your ads relatively to the time when a user is connected to the stream. In case of “stream” value, start time is defined in seconds.
In this case “time_offset" will set when to start the ad after a user is connected. If it’s “time_offset":"0" then it’s inserted right after the user has connected.

The “**time_interval**" element defined the time period for next ads insertions. It’s defined in seconds. E.g. if it’s “3600" this means it will be played every hour. If it’s “0" thet it will not be repeated.

The combination of these 3 elements allows defining different timing scenarios.
If you need to define the pre-roll ad, you will use
“time_sync":"stream",“time_offfset":"0","time_interval":"0"
If you need to start ads 5 minutes before the end of each hour, you will use
“time_sync":"gmt",“time_offfset":"2018-01-01 00:55:00","time_interval":"3600"

The “**contents**" element defines a list of elements, each describing one ad to insert at the time point defined by previous elements.
Each element has “id" which is the ID of an ad described in starting “contents" element of handler response.

The “onerror" element defined what to do if the ad is not yet in Nimble Streamer cache. Possible values are “stop" or "skip": “stop" means Advertizer will stop the original stream for this user and “skip" means Advertizer will continue playing original stream without an ad.
The “wait" element defined time in seconds which Nimble Advertizer will wait until the ad in downloaded.
So "id":"11","onerror":"stop","wait":"10" means Nimble will wait for 10 seconds trying to download the “11" ad and will stop the streaming if it fails to download during that time.
And "id":"22","onerror":"skip","wait":"2" means Nimble will wait for 2 seconds while it downloads “22" ad and if it doesn’t get the ad, then the playback will resume.

This is basic grammar covering major use cases. Let’s see some examples of responses for typical situations.

6. Some examples

The following response sets Nimble Advertizer to play 2 ads.

1. At the beginning of each hour for all streams on the server.
2. Pre-roll for stream radio/stream1 on this server for every connected user.

```
{
  "contents":
  [
    {"id":"1","uri":"http:\/\/192.168.232.129\/main_sponsor.aac.mp4"},
    {"id":"2","uri":"http:\/\/192.168.232.129\/pre-roll.aac.mp4"}
  ],
  "rules":
  [
    {
    "id":"1",
    "protocols":["icecast", "hls"],
    "users":[],
    "type":"global",
    "time_sync":"gmt",
    "time_offset":"2018-01-01 00:00:00",
    "time_interval":"3600",
    "contents":[{"id":"1","onerror":"skip","wait":"2"}
    },
    {
    "id":"2",
    "protocols":["icecast"],
    "users":[],
    "type":"stream",
    "app":"radio",
    "stream":"stream1",
    "time_sync":"stream",
    "time_offset":"0",
    "time_interval":"0",
    "contents":[{"id":"2","onerror":"stop","wait":"10"}
    }
  ]
}
```

Another example is the emulation of “big red button” push: your stream is on air and you want to insert new ad right now. Let’s assume it’s January, 15th 2018 and it’s 20:15 GMT. Your server advertising_sync_interval parameter is set to “2” to allow quick changes. Your handler response will be as follows to run the ad in a few seconds.

```
{
  "contents":[{"id":"1","uri":"http:\/\/192.168.232.129\/new_ad.aac.mp4"}],
  "rules":
  [
    {
    "id":"1",
    "protocols":["icecast"],
    "users":[],
    "type":"global",
    "time_sync":"gmt",
    "time_offset":"2018-15-01 20:15:05",
    "time_interval":"3600",
    "contents":[{"id":"1","onerror":"skip","wait":"10"}
    }
  ]
}
```

In this case as soon as you send new response in the nearest sync, Advertizer will start the download of new ad. In 20:15:05 Advertizer will try to play an ad. If the ad file is in cache, it will be played. If the ad is still being downloaded, Advertizer will wait for 10 seconds (which should be enough to download small audio file) and then all radio listeners will hear this ad.

7. Ads logging for HLS

We plan adding extended logging for ads insertion. For how you can track HLS ads insertion via access.log file. Here’s an example of such logging:

```
127.0.0.1 - - [2018-07-30 05:49:38] "GET
/live/stream/ad_50_6_0_0.ts?nimblesessionid=1 HTTP/1.1" 200 400 2821398 11052
"HTTP Referrer" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/68.0.3440.75 Safari/537.36"
127.0.0.1 - - [2018-07-30 05:49:38] "GET
/live/stream/ad_50_6_1_1.ts?nimblesessionid=1 HTTP/1.1" 200 400 810925 10613
```

```
"HTTP Referrer" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/68.0.34
```

Here you can see each chunk access being logged, having standard log format. The chunk name is as follows: `ad_<content_ID>_<chunk duration>_<chunk_index>_<unique_index>.ts`
The “ad_” prefix for a chunk means it’s an advertizement content, then it has content ID as defined in handler’s response and desired chunk duration. Chunk index and unique index are used for identification of chunks within session.

If you have any questions on Nimble Advertizer usage, please contact us via company helpdesk - https://wmspanel.com/help.