Centralized server data logging to ClickHouse DB

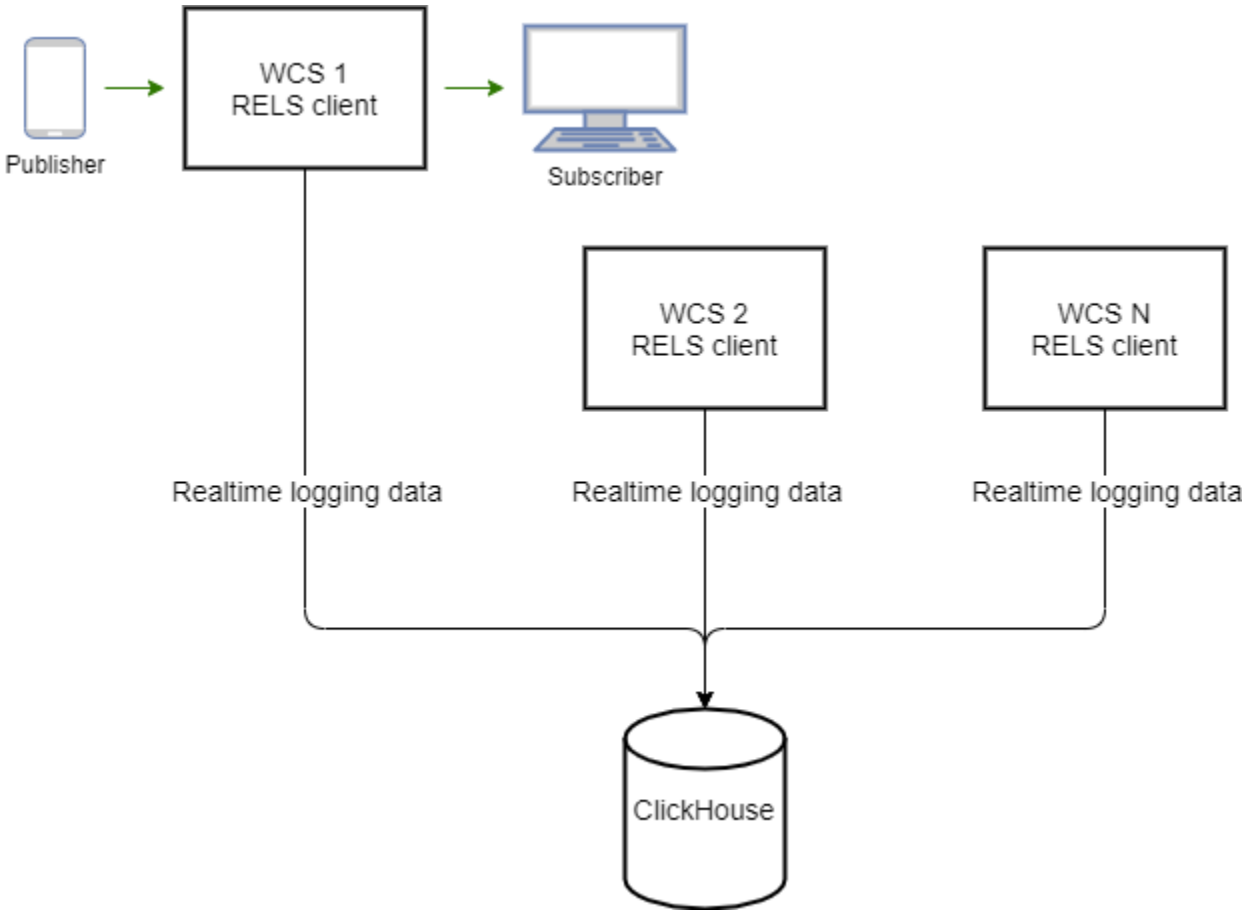
Overview

It may be necessary to collect stream, client connections and CDN events data while managing a big number of WCS servers, to debug a streaming problems. In fact, the information that logged on every server, should be collected at one point. Note that logging itself is minimized in production use, to prevent server disk excessive load.

To collect such big amount of data, time series databases are good choice. Since build5.2.774,the Remote Event Logging System (RELS) based on open source time series DBClickHousecan be used to collect server logs.

Architecture

Every WCS server sends logging data to ClickHouse DB independently using JDBC-driver and HTTP connection. To optimize ClickHouse server load, the data are buffered and sent by time interval batch



Data table description

The logging data are collected to ClickHouse tables listed below. To speed up, an integer event identifiers are written to the tables. There is a textual string dictionary decribing all the events for each table to display human readable selection results.

Connections data (table ConnectionEvent)

Field	Type	Description
timestamp	UInt64	Time stamp
ip	IPv4	Server address
sessionId	String	Session Id
eventType	UInt64	Event type Id
eventPayload	String	Event payload

Streams data (table StreamEvent)

Field	Type	Description
timestamp	UInt64	Time stamp
ip	IPv4	Server address
sessionId	String	Session Id
mediaSessionId	String	Media session Id
streamName	String	Stream name
eventType	UInt64	Event type Id

eventPayload	String	Event payload

CDNdata (table CDNEvent)

Field	Type	Description
timestamp	UInt64	Time stamp
ip	IPv4	Server address
nodeId	String	Node Id (IP address string)
eventType	UInt64	Event type Id
eventPayload	String	Event payload

Stream metrics data (table MediaSessionEvents)

Since build 5.2.1896 it is possible to collect a certain stream metrics data

Field	Type	Description
timestamp	UInt64	Time stamp
ip	IPv4	Server address
mediaSessionId	String	Media session Id
streamName	String	Stream name
videoProfileId	UInt32	Video profile Id
videoWidth	UInt32	Pictiure height
videoHeight	UInt32	Picture width
videoFrameRate	UInt32	Video frame rate per second
videoKframes	UInt64	Key frames number (I-frame)
videoPframes	UInt64	P-frames number
videoBframes	UInt64	B-frames number
videoRate	UInt64	Video bitrate, bps
audioRate	UInt64	Audio bitrate, bps
videoSyncTime	UInt64	Video synchronization value
audioSyncTime	UInt64	Audio synchronization value
videoTimestamp	UInt64	Video packet timestamp
audioTimestamp	UInt64	Audio packet timestamp
lastKeyFrameSyncTime	UInt64	Video synchronization value from the last key frame
sendNACK	UInt64	NACK sent count
recvNACK	UInt64	NACK received count
videoFramesLost	UInt64	Lost video frames count
audioPacketsLost	UInt64	Lost audio frames count
audioPlaybackSpeed	Float32	Audio publishing/playback speed
videoPlaybackSpeed	Float32	Video publishing/playback speed

HLS stream data (table HlsStreamEvents)

Since build 5.2.1917 it is possible to collect a certain HLS stream events data

Field	Type	Description

timestamp	UInt64	Time stamp
ip	IPv4	Server address
severity	UInt8	Event severity level: INFO, WARNING, ERROR
messageType	UInt16	Event type
streamId	String	HLS stream Id
variantName	String	Quality variant name
segmentId	String	HLS segment Id
message	String	Logged event description

HLS segments data (table HlsSegmenterEvents)

Since build 5.2.1917 it is possible to collect a certain HLS stream segments data

Field	Type	Description
timestamp	UInt64	Time stamp
ip	IPv4	Server address
streamId	String	HLS stream Id
variantName	String	Quality variant name
segmentId	String	Segment Id
segmentStartPts	UInt64	Segment start PTS
videoStartPts	UInt64	Video start PTS
audioStartPts	UInt64	Audio start PTS
videoWidth	UInt32	Picture width
videoHeight	UInt32	Picture height
videoFrameCount	UInt32	Video frames count
audioPacketCount	UInt32	Audio packets count
segmentDuration	UInt64	Segment duration, ms
independent	Bool	Independent segment
gap	Bool	GAP segment
discontinuity	Bool	DISCONTINUTY segment
segmentInterval	UInt64	Segments interval, ms
partial	Bool	Partial segment
playbackSpeed	Float32	Playback speed

HLS subscribers data (table HlsClientEvents)

Since build 5.2.1929 it is possible to collect a certain HLS stream subscribers data

Field	Type	Description
creationTime	UInt64	Subscriber creation time (a first request)
responseTime	UInt64	Subscriber response time
streamId	String	HLS stream Id
variantName	String	Quality variant name
uri	String	Playlist URI
localIp	IPv4	Server address

remoteIp	IPv4	Client address
remotePort	UInt32	Client port
userAgent	String	User-Agent data
httpStatus	UInt32	Response status
clientId	UInt64	Client session Id

Mixer metrics data (table MixerEvent)

Since build 5.2.1923 it is possible to collect a certain mixer metrics data

Field	Type	Description
timestamp	UInt64	Time stamp
mixerMediaSessionId	String	Mixer media session Id
mixerStreamName	String	Mixer output stream name
mediaSessionId	String	Incoming stream media session Id
streamName	String	Incoming stream name
mixerAverageTickTimeInMs	Int64	Mixer tick average duration, ms
audioMixerSync	Int64	Mixer output stream audio synchronization value
videoMixerSync	Int64	Mixer output stream audio synchronization value
nextAudioDataTime	Int64	Next audio packet time
nextVideoDataTime	Int64	Next video packet time
audioBuffered	Int64	Incoming stream audio packets buffered count
videoBuffered	Int64	Incoming stream video packets buffered count
audioDropsCounter	Int64	Incoming stream audio packets dropped count
audioDropsSizeInBytes	Int64	Incoming stream audio dropped data in bytes
videoDropsCounter	Int64	Incoming stream video packets dropped count
videoDropsSizeInBytes	Int64	Incoming stream video dropped data in bytes
videoFps	Int64	Mixer output stream framerate per second
audioRate	DOUBLE	Mixer output stream audio bitrate, bps
videoRate	DOUBLE	Mixer output stream audio bitrate, bps
eventType	UInt32	Mixer event type
eventPayload	String	Event description

Incoming audio recovery data (table AudioRecoveryEvent)

Since build 5.2.1969 it is possible to collect a certain stream audio recovery data

Field	Type	Description
timestamp	UInt64	Time stamp
mediaSessionId	String	Media session Id
type	UInt32	Event type
rtpTimestamp	UInt64	RTP stream timestamp

RTMP incoming streams buffer metrics data (table RtmpInBufferEvent)

Since build 5.2.1978 it is possible to collect a certain incoming RTMP stream buffer metrics data

Field	Type	Description
timestamp	UInt64	Time stamp
streamClockTime	UInt64	Stream clock time
mediaSessionId	String	Media session Id
streamName	String	Stream name
nextAudioDataTime	Int64	Next audio data packet time
nextVideoDataTime	Int64	Next video data packet time
audioBuffered	Int64	Audio packets buffered count
videoBuffered	Int64	Video packets buffered count
maximumAllowedBuffer	Int64	Maximum packet allowed in the buffer
bufferingCounter	Int64	Bufferings counter
lastAudioDataTime	Int64	Last audio data packet time
lastVideoDataTime	Int64	Last video data packet time
bufferState	UInt32	Buffer state

Configuration

ClickHouse installation and setup

Server requirement

- CPU from 4 physical cores, frequency from 3 GHz, for example Intel(R) Xeon(R) CPU E3-1246 v3 @ 3.50GHz
- RAM from 32 Gb
- HDD from 2 Tb

ClickHouse installation in CentOS 7

1. Create repository file altinity_clickhouse.repo in /etc/yum.repos.d folder

```
sudo cat <<EOF > /etc/yum.repos.d/altinity_clickhouse.repo
[altinity_clickhouse]
name=altinity_clickhouse
baseurl=https://packagecloud.io/altinity/clickhouse/el/7/$basearch
repo_gpgcheck=1
gpgcheck=0
enabled=1
gpgkey=https://packagecloud.io/altinity/clickhouse/gpgkey
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300

[altinity_clickhouse-source]
name=altinity_clickhouse-source
baseurl=https://packagecloud.io/altinity/clickhouse/el/7/SRPMS
repo_gpgcheck=1
gpgcheck=0
enabled=1
gpgkey=https://packagecloud.io/altinity/clickhouse/gpgkey
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300
EOF
```

2. Add repository

```
sudo yum -q makecache -y --enablerepo='altinity_clickhouse'
```

3. Install ClickHouse

```
sudo yum install -y clickhouse-server clickhouse-client
```

4. Launch ClickHouse

```
systemctl start clickhouse-server
```

ClickHouse configuration

1. Uncomment the following string in /etc/clickhouse-server/config.xml file to listen all the server network interfaces

```
    <listen_host>::</listen_host>
```

2. Set the following parameter for default user in /etc/clickhouse-server/users.xml file to temporary allow users managent

```
            <access_management>1</access_management>
```

3. Restart ClickHouse

```
systemctl restart clickhouse-server
```

4. Create wcs database and tables

```
cat wcs_clickhouse.sql | clickhouse-client -mn
```

wcs_clickhouse.sql

```
CREATE DATABASE IF NOT EXISTS wcs;

DROP DICTIONARY IF EXISTS wcs.DictionaryStreamEvents;

DROP DICTIONARY IF EXISTS wcs.DictionaryConnectionEvents;

DROP DICTIONARY IF EXISTS wcs.DictionaryCDNEvents;

DROP DICTIONARY IF EXISTS wcs.DictionaryHlsStreamEventType;

DROP DICTIONARY IF EXISTS wcs.DictionaryHlsStreamEventSeverity;

DROP DICTIONARY IF EXISTS wcs.DictionaryMixerEvents;

DROP DICTIONARY IF EXISTS wcs.DictionaryBufferStateTypes;

DROP TABLE IF EXISTS wcs.StreamEvent;

DROP TABLE IF EXISTS wcs.ConnectionEvent;

DROP TABLE IF EXISTS wcs.CDNEvent;

DROP TABLE IF EXISTS wcs.StreamEventTypes;

DROP TABLE IF EXISTS wcs.ConnectionEventTypes;

DROP TABLE IF EXISTS wcs.CDNEventTypes;

DROP TABLE IF EXISTS wcs.MediaSessionEvents;
```

```
DROP TABLE IF EXISTS wcs.HlsStreamEvents;

DROP TABLE IF EXISTS wcs.HlsSegmenterEvents;

DROP TABLE IF EXISTS wcs.HlsStreamEventSeverity;

DROP TABLE IF EXISTS wcs.HlsStreamEventType;

DROP TABLE IF EXISTS wcs.HlsClientEvents;

DROP TABLE IF EXISTS wcs.MixerEvent;

DROP TABLE IF EXISTS wcs.MixerEventTypes;

DROP TABLE IF EXISTS wcs.RtmpInBufferEvent;

DROP TABLE IF EXISTS wcs.AudioRecoveryEvent;

DROP TABLE IF EXISTS wcs.BufferStateTypes;

CREATE TABLE wcs.ConnectionEventTypes
(
    `id` UInt32,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.ConnectionEventTypes VALUES (0, 'CONNECTED'), (1, 'DISCONNECTED');

CREATE TABLE wcs.StreamEventTypes
(
    `id` UInt32,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.StreamEventTypes VALUES (0,'CREATED'),(1,'LOCAL_SDP_CREATED'),(2,'REMOTE_SDP_RECEIVED'),
(3,'ICE_STARTED'),(4,'ICE_COMPLETE'),(5,'DTLS_STARTED'),(6,'DTLS_COMPLETE'),(7,'INITIALIZED'),(8,'DISPOSING'),
(9,'DISPOSED'),(10,'AUDIO_RECEIVED'),(11,'VIDEO_RECEIVED'),(12,'VIDEO_KFRAME_RECEIVED'),
(13,'AUDIO_RTCP_RECEIVED'),(14,'VIDEO_RTCP_RECEIVED'),(15,'RESOLUTION_RECEIVED'),(16,'VIDEO_ENCODER_CREATED'),
(17,'AUDIO_ENCODER_CREATED'),(18,'VIDEO_ENCODER_DISPOSED'),(19,'AUDIO_ENCODER_DISPOSED'),(20,'TERMINATED'),
(21,'AUDIO_SENT'),(22,'VIDEO_SENT'),(23,'VIDEO_JITTER_BUFFER_STALL'),(24,'SENT_PLI'),(25,'RECEIVED_PLI'),
(26,'SYNC_BUFFER_FULL'),(27,'SYNC_FORCE_FAILED'),(28,'SYNC_SHIFT'),(29,'SYNC_DEVIATION'),(30,'VIDEO_STATS'),
(31,'RECORD');

CREATE TABLE wcs.CDNEventTypes
(
    `id` UInt32,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.CDNEventTypes VALUES (0, 'STATE'), (1, 'CDN_STATE'), (2, 'VERSION'), (3, 'ACL_REFRESH'), (4,
'ACL_UPDATE');

CREATE DICTIONARY wcs.DictionaryStreamEvents (
    `id` UInt16,
    `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
```

```
 password ''
 db 'wcs'
 table 'StreamEventTypes'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE DICTIONARY wcs.DictionaryConnectionEvents (
    `id` UInt16,
    `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'ConnectionEventTypes'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE DICTIONARY wcs.DictionaryCDNEvents (
    `id` UInt16,
    `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'CDNEventTypes'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE TABLE wcs.StreamEvent
(
    `timestamp` UInt64,
    `ip` IPv4,
    `sessionId` String,
    `mediaSessionId` String,
    `streamName` String,
    `eventType` UInt64,
    `eventPayload` String
)
ENGINE = MergeTree()
ORDER BY (sessionId, mediaSessionId, streamName)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.ConnectionEvent
(
    `timestamp` UInt64,
    `ip` IPv4,
    `sessionId` String,
    `eventType` UInt64,
    `eventPayload` String
)
ENGINE = MergeTree()
ORDER BY (timestamp, sessionId)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.CDNEvent
(
    `timestamp` UInt64,
    `ip` IPv4,
    `nodeId` String,
    `eventType` UInt64,
```

```
    `eventPayload` String
)
ENGINE = MergeTree()
ORDER BY (nodeId, eventType)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.MediaSessionEvents
(
    `timestamp` UInt64,
    `ip` IPv4,
    `mediaSessionId` String,
    `streamName` String,
    `videoProfileId` UInt32,
    `videoWidth` UInt32,
    `videoHeight` UInt32,
    `videoFrameRate` UInt32,
    `videoKframes` UInt64,
    `videoPframes` UInt64,
    `videoBframes` UInt64,
    `videoRate` UInt64,
    `audioRate` UInt64,
    `videoSyncTime` UInt64,
    `audioSyncTime` UInt64,
    `videoTimestamp` UInt64,
    `audioTimestamp` UInt64,
    `lastKeyFrameSyncTime` UInt64,
    `sendNACK` UInt64,
    `recvNACK` UInt64,
    `videoFramesLost` UInt64,
    `audioPacketsLost` UInt64,
    `audioPlaybackSpeed` Float32,
    `videoPlaybackSpeed` Float32
)
ENGINE = MergeTree()
ORDER BY (mediaSessionId, streamName)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.HlsSegmenterEvents
(
    `timestamp` UInt64,
    `ip` IPv4,
    `streamId` String,
    `variantName` String,
    `segmentId` String,
    `segmentStartPts` UInt64,
    `videoStartPts` UInt64,
    `audioStartPts` UInt64,
    `videoWidth` UInt32,
    `videoHeight` UInt32,
    `videoFrameCount` UInt32,
    `audioPacketCount` UInt32,
    `segmentDuration` UInt64,
    `independent` Bool,
    `gap` Bool,
    `discontinuity` Bool,
    `segmentInterval` UInt64,
    `partial` Bool,
    `playbackSpeed` Float32
)
ENGINE = MergeTree()
ORDER BY (streamId)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.HlsStreamEventSeverity
(
    `id` UInt8,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;
```

```
INSERT INTO wcs.HlsStreamEventSeverity VALUES (0, 'INFO'), (1, 'WARNING'), (2, 'ERROR');

CREATE TABLE wcs.HlsStreamEventType
(
    `id` UInt16,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.HlsStreamEventType
VALUES (0, 'PLAYBACK_SPEED'), (1, 'FPS_CHANGED'), (2, 'GAP'), (3, 'RESOLUTION_CHANGED'), (4, 'DISCONTINUITY'),
(5, 'TASK_SKIPPED'), (6, 'NO_KYE_FRAME'), (7, 'NO_VIDEO'), (8, 'NO_AUDIO'), (9, 'SEGMENT_INTERVAL'), (10,
'OTHER');

CREATE DICTIONARY wcs.DictionaryHlsStreamEventSeverity
(
   `id` UInt8,
   `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'HlsStreamEventSeverity'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE DICTIONARY wcs.DictionaryHlsStreamEventType
(
   `id` UInt16,
   `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'HlsStreamEventType'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE TABLE wcs.HlsStreamEvents
(
    `timestamp` UInt64,
    `ip` IPv4,
    `severity` UInt8,
    `messageType` UInt16,
    `streamId` String,
    `variantName` String,
    `segmentId` String,
    `message` String
)
ENGINE = MergeTree()
ORDER BY (streamId)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.HlsClientEvents
(
    `creationTime` UInt64,
    `responseTime` UInt64,
    `streamId` String,
```

```
    `variantName` String,
    `uri` String,
    `localIp` IPv4,
    `remoteIp` IPv4,
    `remotePort` UInt32,
    `userAgent` String,
    `httpStatus` UInt32,
    `clientId` UInt64
)
ENGINE = MergeTree()
ORDER BY (creationTime)
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.MixerEvent
(
    `timestamp` UInt64,
    `mixerMediaSessionId` String,
    `mixerStreamName` String,
    `mediaSessionId` String,
    `streamName` String,
    `mixerAverageTickTimeInMs` Int64,
    `audioMixerSync` Int64,
    `videoMixerSync` Int64,
    `nextAudioDataTime` Int64,
    `nextVideoDataTime` Int64,
    `audioBuffered` Int64,
    `videoBuffered` Int64,
    `audioDropsCounter` Int64,
    `audioDropsSizeInBytes` Int64,
    `videoDropsCounter` Int64,
    `videoDropsSizeInBytes` Int64,
    `videoFps` Int64,
    `audioRate` DOUBLE,
    `videoRate` DOUBLE,
    `eventType` UInt32,
    `eventPayload` String
)
ENGINE = MergeTree()
ORDER BY timestamp
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.MixerEventTypes
(
    `id` UInt32,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.MixerEventTypes VALUES (0, 'nullEvent'), (1, 'dropBallastAudio'), (2, 'dropBallastVideo'), (3,
'audioNotBuffered'), (4, 'videoNotBuffered'), (5, 'audioBufferExhausted'), (6, 'videoBufferExhausted'), (7,
'alignStreamFailed'), (8, 'alignStreamDropAudio'), (9, 'alignStreamDropVideo'), (10, 'rateOutOfBoundsAudio'),
(11, 'rateOutOfBoundsVideo');

CREATE DICTIONARY wcs.DictionaryMixerEvents (
    `id` UInt16,
    `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'MixerEventTypes'
))
LAYOUT(FLAT())
LIFETIME(300);
```

```
CREATE TABLE wcs.RtmpInBufferEvent
(
    `timestamp` UInt64,
    `streamClockTime` UInt64,
    `mediaSessionId` String,
    `streamName` String,
    `nextAudioDataTime` Int64,
    `nextVideoDataTime` Int64,
    `audioBuffered` Int64,
    `videoBuffered` Int64,
    `maximumAllowedBuffer` Int64,
    `bufferingCounter` Int64,
    `lastAudioDataTime` Int64,
    `lastVideoDataTime` Int64,
    `bufferState` UInt32
)
ENGINE = MergeTree()
ORDER BY timestamp
SETTINGS index_granularity = 8192;

CREATE TABLE wcs.BufferStateTypes
(
    `id` UInt32,
    `type` String
)
ENGINE = MergeTree()
ORDER BY id
SETTINGS index_granularity = 8192;

INSERT INTO wcs.BufferStateTypes VALUES (0, 'BUFFERING'), (1, 'HOLD'), (2, 'TERMINATED'), (3, 'OVERFLOW'), (4,
'PASSTHROUGH');

CREATE DICTIONARY wcs.DictionaryBufferStateTypes (
    `id` UInt16,
    `type` String DEFAULT ''
)
PRIMARY KEY id
SOURCE(CLICKHOUSE(
 host 'localhost'
 port 9000
 user 'default'
 password ''
 db 'wcs'
 table 'BufferStateTypes'
))
LAYOUT(FLAT())
LIFETIME(300);

CREATE TABLE wcs.AudioRecoveryEvent
(
    `timestamp` UInt64,
    `mediaSessionId` String,
    `type` UInt32,
    `rtpTimestamp` UInt64
)
ENGINE = MergeTree()
ORDER BY rtpTimestamp
SETTINGS index_granularity = 8192;
```

5.Create wcs user and grant permissions to all the tables in wcs database

```
cat wcs_clickhouse_users.sql | clickhouse-client -mn
```

wcs_clickhouse_users.sql

```
CREATE USER IF NOT EXISTS wcs IDENTIFIED BY 'wcs';
GRANT ALL ON wcs.* TO wcs WITH GRANT OPTION;
```

6.Disable users management for default user by setting the following parameter in /etc/clickhouse-server/users.xml file

```
            <access_management>0</access_management>
```

7. Restart ClickHouse

```
systemctl restart clickhouse-server
```

WCS configuration

Data logging to ClickHouse is enabled by the following list of the data to collect

```
rels_enabled=CONNECTION,STREAM,CDN,MEDIA_SESSION
```

The following data types are available:

CONNECTION, STREAM, CDN, MEDIA_SESSION, HLS_SEGMENTER, HLS_STREAM, HLS_CLIENT, MIXER, AUDIO_RECOVERY, RTMP_IN_BUFFER

Type	Description
CONNECTION	Client session events
STREAM	Stream events
CDN	CDN events
MEDIA_SESSION	Stream metrics
HLS_SEGMENTER	HLS stream segment metrics
HLS_STREAM	HLS stream events
HLS_CLIENT	HLS clients statistics
MIXER	Mixer events and metrics
AUDIO_RECOVERY	Audio packets recovery events
RTMP_IN_BUFFER	RTMP incoming streams buffer metrics

ClickHouse server, database address and protocol are set by the following parameters

```
rels_client_type=HTTP
rels_database_address=http://clickhouseserver:8123/wcs?user=wcs&password=wcs
```

HTTP protocol is recommended and used by default. But it can be switched to JDBC driver if needed

```
rels_client_type=JDBC
rels_database_address=jdbc:clickhouse://clickhouseserver:8123/wcs?user=wcs&password=wcs
```

Stop data logging without WCS server restart

Data logging can be stopped without WCS restart if necessary. To do this:

1. Disable data logging in server settings

```
rels_enabled=false
```

2.Reload settings usingCLI command

```
reload node-settings
```

ClickHouse server address changingwithout WCS server restart

ClickHouse server address can be changedwithout WCS restart. To do this:

1. Change address in server settings

```
rels_database_address=jdbc:clickhouse://newclickhouseserver:8123/wcs?user=wcs&password=wcs
```

2.Disable data logging in server settings

```
rels_enabled=false
```

3.Reload settings usingCLI command

```
reload node-settings
```

4. Enable data logging in server settings

```
rels_enabled=true
```

5.Reload settings usingCLI command

```
reload node-settings
```

Data collection management using REST API

The data of CONNECTION, STREAM, CDN, HLS_STREAM types are always collected, for all the client sessions and all the streams. All the other types are collected on demand only because a data amount may be too big.

The data collection for a certain stream is enabled by REST API.

A REST-query should be HTTP/HTTPS POST request as follows:

- HTTP:http://streaming.flashphoner.com:8081/rest-api/rels/startup
- HTTPS:https://streaming.flashphoner.com:8444/rest-api/rels/startup

Здесь:

- streaming.flashphoner.com - WCS server address
- 8081 - the standard REST / HTTP port of the WCS server
- 8444 - the standard HTTPS port
- rest-api - the required part of the URL
- /rels/startup - REST-method to use

REST methods and responses

/rels/startup

Start data collectio of a certain type for the certain streams

Request example

```
POST /rest-api/rels/startup HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
    "mediaSession": {
        "frequency":100,
        "ids": [
            "d7d6b6e4-b137-461a-8a32-d6abf2b8666e",
            "39cbf770-128a-11ef-b839-d1a1f53f8bd2"
        ]
    }
}
```

Response example

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

Return codes

Code	Reason
200	OK
400	Bad request
404	Not found
500	Internal server error

/rels/find_all

Get the sent data statistics

Request example

```
POST /rest-api/rels/find_all HTTP/1.1
Host: localhost:8081
Content-Type: application/json
```

Response example

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json

[
  {
    "type": "CONNECTION",
    "sentBytes": 1989,
    "bitrateKbps": 0,
    "sentEvents": 6,
    "queueEvents": 0
  },
  {
    "type": "STREAM",
    "sentBytes": 70766,
    "bitrateKbps": 0,
    "sentEvents": 73,
    "queueEvents": 1
  },
  {
    "type": "CDN",
    "sentBytes": 0,
    "bitrateKbps": 0,
    "sentEvents": 0,
    "queueEvents": 0
  },
  {
    "type": "MEDIA_SESSION",
    "ids": [
      "d7d6b6e4-b137-461a-8a32-d6abf2b8666e",
      "39cbf770-128a-11ef-b839-d1a1f53f8bd2"
    ],
    "sentBytes": 205794,
    "bitrateKbps": 143,
    "sentEvents": 999,
    "queueEvents": 119
  }
]
```

Return codes

Code	Reason
200	OK
404	Not found
500	Internal server error

/rels/terminate

Stop data collection of the certain type for the certain streams

Request example

```
POST /rest-api/rels/terminate HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
    "mediaSession": {
        "ids": [
            "d7d6b6e4-b137-461a-8a32-d6abf2b8666e"
        ]
    }
}
```

Response example

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

Return codes

Code	Reason
200	OK
400	Bad request
404	Not found
500	Internal server error

/rels/terminate_all

Stop the certain types data collection for all the streams

Request example

```
POST /rest-api/rels/terminate_all HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
    "types": [
        "MEDIA_SESSION"
    ]
}
```

Response example

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json

{
  "MEDIA_SESSION": [
    "39cbf770-128a-11ef-b839-d1a1f53f8bd2"
  ]
}
```

Return codes

Code	Reason
200	OK
400	Bad request
404	Not found
500	Internal server error

Parameters

Parameter	Description	Example
mediaSession	MEDIA_SESSION data collection descriptor object	`"mediaSession": {"frequency":100, "ids":["12345678-0000-1111"]}`

frequency	data collection frequency in ms	100
ids	Medissesson identifiers list for the streams which data should be collected	`["12345678-0000-1111","12345678-3333-4444"]`
hlsSegmenter	HLS_SEGMENTER data collection descriptor object	"hlsSegmenter": `{"ids":["stream1"]}`
hlsClient	HLS_CLIENT data collection descriptor object	"hlsClient": `{"ids":["stream1"]}`
mixer	MIXER data collection descriptor object	"mixer": `{"ids":["12345678-5555-6666"]}`
audioRecovery	AUDIO_RECOVERY data collection descriptor object	"audioRecovery": `{"ids":["12345678-7777-8888"]}`
rtmpInBuffer	RTMP_IN_BUFFER data collection descriptor object	"rtmpInBuffer": `{"ids":["12345678-9999-AAAA"]}`

Data retrieving from DB

Logging data can be retrieved usin SQL queries in CliskHouse client

```
select timestamp,ip,sessionId,mediaSessionId,streamName,dictGetString('wcs.DictionaryStreamEvents','type',
eventType) as eventType from wcs.StreamEvent where streamName = 'test'
```

```
select timestamp,ip,sessionId,dictGetString('wcs.DictionaryConnectionEvents','type', eventType) as eventType
from wcs.ConnectionEvent
```

```
select timestamp,ip,nodeId,dictGetString('wcs.DictionaryCDNEvents','type', eventType) as eventType,eventPayload
from wcs.CDNEvent
```