Ceeblue Streaming Cloud API Guide

Ceeblue Cloud Dev Team

Version 0.4.5, 8 Feb 2021

Table of Contents

Changes . . . 1
Logging In . . . 2
Authorization . . . 3
1. Overview . . . 4
2. HTTP verbs . . . 5
3. HTTP status codes . . . 6
4. Streams . . . 7
4.1. Input stream . . . 7
4.1.1. Creating input . . . 7
4.1.2. Retrieving streams . . . 14
4.1.3. Fetching stream . . . 15
4.2. Update stream . . . 15
4.2.1. Deleting stream . . . 16
4.2.2. Fetching output settings . . . 16
4.2.3. Update a output settings . . . 17
4.3. Output stream . . . 17
4.3.1. Request output . . . 17
4.3.2. Retrieving outputs by stream id . . . 19
4.3.3. Output sessions invalidation . . . 20
4.3.4. Deleting output . . . 20
5. Stream push . . . 21
5.1. Add push . . . 21
5.2. Fetching stream push . . . 23
5.3. Deleting stream . . . 24
5.4. Retrieving pushes by stream id . . . 24
6. Stream recording . . . 25
6.1. Add recording . . . 25
6.2. Fetching recording . . . 28
6.3. Fetching recordings . . . 29
6.4. Fetching recordings by stream id . . . 29
6.5. Stop recording . . . 30
6.6. Deleting recording . . . 30
7. Stream snapshot . . . 32
7.1. Fetching stream snapshot . . . 32
7.2. Setup snapshot settings . . . 32
7.3. Fetching snapshot settings . . . 33
7.4. Clear snapshot settings . . . 33
8. Stream information . . . 34

8.1. Fetching stream information . . . 34
9. Storage . . . 36
9.1. Add storage . . . 36
9.2. Fetching storage . . . 37
9.3. Retrieving storages . . . 38
9.4. Update storage . . . 38
9.5. Delete storage . . . 39
Appendix A: Stream status . . . 40

Changes

- **0.4.5 02 Mar 2021** Add MPEG-TS (UDP) push (Stream push)
- **0.4.4 08 Feb 2021** The ability to customize the transcoding of audio tracks
- **0.4.3 18 Jan 2021** Append **Pending** stream status
- **0.4.2 16 Dec 2020** The Secure Reliable Transport (SRT) protocol ingestion support
- **0.4.1 08 Dec 2020** Added automatic stream pushing, triggers by "Ingestion" stream state

Logging In

```
POST /login
```

Request

Path	Type	Required	Description
username	string	yes	The user unique name
password	string	yes	The user secret

Logging in request example

```
{
    "username": "user",
    "password": "e1c21ed98788f81e7191"
}
```

Response
JWT token or `403 Forbidden`

Success response

```
{
  "token": "eyJ. . .CJ9.eyJ. . .DZ9.Zqh. . .F0U"
}
```

Authorization

When the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally. Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema.

Example

```
curl -v -H "Content-Type: application/json" -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJhNzgzYjhjZC01ODNjLTQ4NWQtOTcyMS1mZ
DczN2RhOGQ0YTIiLCJpYXQiOjE1MjQ1Nzk4MzR9.XdK3-kbL80SnoHvvUNUKEpZg0E3PaN3YYpywv9pUxFg"
http://localhost:3000/inputs
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET /inputs HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.58.0
> Accept: */*
> Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJhNzgzYjhjZC01ODNjLTQ4NWQtOTcyMS1mZ
DczN2RhOGQ0YTIiLCJpYXQiOjE1MjQ1Nzk4MzR9.XdK3-kbL80SnoHvvUNUKEpZg0E3PaN3YYpywv9pUxFg
```

Chapter 1. Overview

The REST API lets you interact with Ceeblue Cloud Server from anything that can send an HTTP request. There are many things you can do with the REST API.

Chapter 2. HTTP verbs

Ceeblue-cloud tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP verbs.

Verb	Usage
`GET`	Used to retrieve a resource
`POST`	Used to create a new resource
`DELETE`	Used to delete an existing resource

Chapter 3. HTTP status codes

RESTful Contracts tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP status codes.

Status code	Usage
`200 OK`	Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.
`201 Created`	The request has been fulfilled and resulted in a new resource being created.
`400 Bad Request`	The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).
`404 Not Found`	The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.

Chapter 4. Streams

4.1. Input stream

4.1.1. Creating input

To create a new input stream on Ceeblue Cloud, send a POST request to the class URL containing the contents of the input

```
POST /inputs
```

Request

Path	Type	Required	Description
format	**enum** string	yes	Input stream format [**RTMP** | **RTSP** | **WebRTC** | **SRT**]
streamer	**InputStreamer**	no	Streamer details
callbackUri	string	no	Invokes on status change. (HTTP POST)
output	**Output**	no	Stream output settings
access	**enum** string	no	Access mode [**Public** | **Private**] (Default: **Public**)
accessToken	string	no	Access token. Will be generated automatically if empty. (**Private** access mode)

Table 1. InputStreamer

Path	Type	Required	Description
ipAddress	string	no	IP address
geoLocation	**GeoLocation**	no	Geographical location

Table 2. GeoLocation

Path	Type	Required	Description
latitude	number	yes	Latitude coordinate in decimal degree format
longitude	number	yes	Longitude coordinate in decimal degree format

Table 3. Output

Path	Type	Required	Description
passthrough	boolean	no	If true, enables a signal to "pass through" unaltered. If false, enables multiprofile transcoding.

Path	Type	Required	Description
version	number	no	Output configuration version [**1** | **2**] (Version **1** by default).
overlay	**Overlay**	no	Overlays an picture onto a video track
tracks	array of **Track**	no (ver. **1**); yes (ver. **2**)	List of video (and audio tracks for the version **2**) transcoding tracks

Version 1: Supports: overlay, video tracks transcoding. If overlay and tracks are empty, the source video tracks will be passthrough. If overlay not empty and tracks are empty, the H264 and VP8 video tracks (with default settings) will be used.

Version 2: Supports: overlay, video and audio tracks transcoding.

Table 4. Overlay

Path	Type	Required	Description
data	string	no	The data URI image.
offsetX	number	no	Horizontal offset of overlay image in pixels from left of video image. For negative value, horizontal offset of overlay image in pixels from right of video image.
offsetY	number	no	Vertical offset of overlay image in pixels from top of video image. For negative value, vertical offset of overlay image in pixels from bottom of video image.

Table 5. Video track

Path	Type	Required	Description
type	enum string	yes	Track type [**Video**]
width	number	no	The width of the output (Min 2)
height	number	no	The height of the output (Min 2)
framerate	number	no	The frame rate of the output
settings	object	yes	Encoder settings **H264 Encoder Settings**, **VP8 Encoder Settings**

Table 6. Audio track

Path	Type	Required	Description
type	enum string	yes	Track type [**Audio**]
rate	number	no	The sample rate output (8000-192000)
channels	number	no	The output number of channels (1-2)

Path	Type	Required	Description
settings	object	yes	Encoder settings **AAC Encoder Settings**, **Opus Encoder Settings**, **MP3 Encoder Settings**

Opus supports 8000, 12000, 16000, 24000, 48000 sample rates

AAC supports 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000, 64000, 88200, 96000 sample rates

MP3 supports 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000 sample rates

Table 7. H264 Encoder Settings

Path	Type	Required	Description
codec	const string	yes	Codec name [**H264**]
bitrate	number	yes	Bitrate in kbit/sec (Min 100)
speedPreset	**enum** string	yes	Preset name for speed/quality tradeoff options [**ultrafast** | **superfast** | **veryfast** | **faster** | **fast** | **medium**]
keyIntMax	number	yes	Maximal distance between two key-frames (Min 2)

Table 8. VP8 Encoder Settings

Path	Type	Required	Description
codec	const string	yes	Codec name [**VP8**]
bitrate	number	yes	Bitrate in kbit/sec (Min 100)
keyMaxDist	number	yes	Maximal distance between two key-frames (Min 2)

Table 9. AAC Encoder Settings

Path	Type	Required	Description
codec	const string	yes	Codec name [**AAC**]
bitrate	number	yes	Bitrate in kbit/sec (Min 10)

Table 10. Opus Encoder Settings

Path	Type	Required	Description
codec	const string	yes	Codec name [**Opus**]
bitrate	number	yes	Bitrate in kbit/sec (Min 10)

Table 11. MP3 Encoder Settings

Path	Type	Required	Description
codec	const string	yes	Codec name [**MP3**]
bitrate	number	yes	Bitrate in kbit/sec (Min 10)

Table 12. Secure Reliable Transport protocol (query string settings)

Path	Type	Default	Description
conntimeo	number	3000	Connect timeout. Milliseconds to wait before timing out a connection attempt for caller and rendezvous modes.
enforcedencryption	boolean	true	Enforced Encryption. If enabled, enforces that both sides either set no passphrase, or set the same passphrase. When disabled, falls back to no passphrase if the passphrases do not match. (0-False/1-True)
inputbw	number	0	Input bandwidth. Estimated bandwidth of data to be sent. (0 means automatic)
passphrase	string	*empty*	Enables encryption with the given passphrase.
pbkeylen	number	0	Encryption key length [**0 (auto)** | **16 (AES-128)** | **24 (AES-192)** | **32 (AES-256)**]

Example request

```
{
    "format": "RTMP",
    "streamer": {
        "ipAddress": "101.69.101.56"
    },
    "callbackUri": "http://mydomain.com/notifications"
}
```

Example request with watermark

```
{
  "format": "RTMP",
  "output": {
    "passthrough": false,
    "overlay": {
      "data": "...AAElFTkSuQmCC"
    },
    "tracks": []
  }
}
```

Example request with few quality transcoding

```
{
  "format": "RTMP",
  "output": {
    "passthrough": false,
    "overlay": {
      "data": "...kDYbvk",
      "offsetX": -15,
      "offsetY": 15
    },
    "tracks": [
      {
        "type": "Video",
        "width": 1920,
        "height": 1080,
        "framerate": 30,
        "settings": {
          "codec": "H264",
          "bitrate": 4000,
          "speedPreset": "superfast",
          "keyIntMax": 30
        }
      }, {
        "type": "Video",
        "width": 1280,
        "height": 720,
        "framerate": 30,
        "settings": {
          "codec": "H264",
          "bitrate": 2000,
          "speedPreset": "superfast",
          "keyIntMax": 30
        }
      }, {
        "type": "Video",
        "width": 960,
        "height": 540,
        "framerate": 30,
        "settings": {
          "codec": "H264",
          "bitrate": 1500,
          "speedPreset": "superfast",
          "keyIntMax": 30
        }
      }, {
        "type": "Video",
        "width": 640,
        "height": 360,
        "framerate": 30,
        "settings": {
          "codec": "H264",
          "bitrate": 1000,
```

```
          "speedPreset": "superfast",
          "keyIntMax": 30
        }
      }, {
        "type": "Video",
        "width": 256,
        "height": 144,
        "framerate": 30,
        "settings": {
          "codec": "H264",
          "bitrate": 500,
          "speedPreset": "superfast",
          "keyIntMax": 30
        }
      }
    ]
  }
}
```

Response

When the creation is successful, the HTTP response is a `201 Created` and the response body is a JSON object containing the stream id, etc.

Path	Type	Description
id	string	Stream identifier
format	**enum** string	Input stream format [**RTMP** | **RTSP** | **WebRTC**]
connection	**HttpConnection** | **WebRTCConnection**	Input endpoint parameters for streaming
streamer	**Streamer**	Streamer details
parameters	object	Input stream parameters
status	**enum** string	**Pending** | **Idle** | **Ingestion** | **Error** (Stream status)
statusMessages	array of string	List of messages
callbackUri	string	Invokes on status change. (HTTP POST)
access	**enum** string	Access mode [**Public** | **Private**]
accessToken	string	Access token. (**Private** access mode)

Table 13. HttpConnection

Path	Type	Description
uri	string	Streaming endpoint

Table 14. WebRTCConnection

Path	Type	Description
signallingUri	string	Streaming endpoint

Path	Type	Description
stun	string	STUN endpoint
turn	string	TURN endpoint

Table 15. Streamer

Path	Type	Description
ipAddress	string	IP address

Example response

```
{
  "id": "2424655f-7150-44f3-97eb-73b4187a650b",
  "format": "RTMP",
  "connection": {
    "uri": "rtmp://in.ceeblue.net/play/ad+2424655f-7150-44f3-97eb-73b4187a650b"
  },
  "status": "Idle",
  "callbackUri": "http://mydomain.com/notifications",
  "access": "Public"
}
```

Example callback notifications

```
{
  "id": "e84f7403-ec80-4e3e-b56b-bdfea431cdfa",
  "format": "RTMP",
  "callbackUri": "https://some.domain.name/callback",
  "connection": {
    "uri": "rtmp://rye.ceeblue.tv/live/as+e84f7403-ec80-4e3e-b56b-bdfea431cdfa"
  },
  "streamer": {
    "ipAddress":"::ffff:195.69.111.69"
  },
  "status": "Ingestion",
  "access": "Private",
  "accessToken": "qqq",
  "timestamp": 1576939156586
}
```

```
{
  "id": "e84f7403-ec80-4e3e-b56b-bdfea431cdfa",
  "format": "RTMP",
  "callbackUri": "https://some.domain.name/callback",
  "connection": {
    "uri": "rtmp://rye.ceeblue.tv/live/as+e84f7403-ec80-4e3e-b56b-bdfea431cdfa"
  },
  "status": "Idle",
  "access": "Private",
  "accessToken": "qqq",
  "timestamp": 1576939276547
}
```

4.1.2. Retrieving streams

You can also retrieve the contents of a stream object by sending a GET request to the `/inputs` URL.

```
GET /inputs
```

The response body is a JSON object containing list of streams

Example response

```
[
  {
    "id": "08189568-3492-4e15-92da-f438eb56797d",
    "format": "RTMP",
    "callbackUri": "http://mydomain.com/notifications",
    "connection": {
      "uri": "rtmp://in.ceeblue.net/play/ad+08189568-3492-4e15-92da-f438eb56797d"
    },
    "status": "Idle",
    "access": "Public"
  },
  {
    "id": "2424655f-7150-44f3-97eb-73b4187a650b",
    "format": "RTMP",
    "callbackUri": "http://mydomain.com/notifications",
    "connection": {
      "uri": "rtmp://in.ceeblue.net/play/ad+2424655f-7150-44f3-97eb-73b4187a650b"
    },
    "status": "Idle",
    "access": "Public"
  }
]
```

4.1.3. Fetching stream

A `GET` request fetches a specific stream by id.

```
GET /inputs/:id
```

The response body is a JSON object

Example response

```
{
  "id": "2424655f-7150-44f3-97eb-73b4187a650b",
  "format": "RTMP",
  "connection": {
    "uri": "rtmp://in.ceeblue.net/play/ad+2424655f-7150-44f3-97eb-73b4187a650b"
  },
  "streamer": {
    "ipAddress": "::ffff:127.0.0.1"
  },
  "status": "Ingestion",
  "callbackUri": "http://mydomain.com/notifications"
}
```

4.2. Update stream

A `PUT` request updates a specific stream by id.

```
PUT /inputs/:id
```

Request

Path	Type	Required	Description
access	**enum** string	no	Access mode [**Public** | **Private**]
accessToken	string	no	Access token (**Private** access mode)

Note: on `Public` to `Private` access mode change, all viewer session will be invalidated.

The response body is a JSON object

Example response

```
{
  "id": "abdb3356-4639-4c7e-bf51-0fb991f0e136",
  "format": "WebRTC",
  "connection": {
    "signallingUri": "wss://some.node.tv:4433/webrtc/in+abdb3356-4639-4c7e-bf51-
0fb991f0e136"
  },
  "streamer": {
    "ipAddress": "......"
  },
  "status": "Ingestion",
  "access": "Private",
  "accessToken": "4c64e33d-f655-46c8-a19a-8ae145f0c615"
}
```

4.2.1. Deleting stream

To delete an stream from the Ceeblue Cloud, send a DELETE request to `/inputs` with id.

```
DELETE /inputs/:id
```

4.2.2. Fetching output settings

A `GET` request fetches a specific output settings by stream id.

```
GET /inputs/:id/output
```

The response body is a JSON object

Example response

```
{
  "passthrough": false,
  "overlay": {
    "data": " ... AAAAAElFTkSuQmCC",
    "offsetX": -10,
    "offsetY": 10
  },
  "tracks": [
    {
      "type": "Video",
      "width": 1920, "height": 1080, "framerate": 30,
      "settings": {
        "codec": "H264", "bitrate": 4000, "speedPreset": "superfast", "keyIntMax": 30
      }
    },
    {
      "type": "Video",
      "width": 1280, "height": 720, "framerate": 30,
      "settings": {
        "codec": "H264", "bitrate": 2500, "speedPreset": "superfast", "keyIntMax": 30
      }
    },
    {
      "type": "Video",
      "width": 640, "height": 480, "framerate": 30,
      "settings": {
        "codec": "H264", "bitrate": 1000, "speedPreset": "superfast", "keyIntMax": 30
      }
    }
  ]
}
```

4.2.3. Update a output settings

A `PUT` request updates output settings by stream id.

```
PUT /inputs/:id/output
```

See **Output**

4.3. Output stream

4.3.1. Request output

When the creation is successful, the HTTP response is a `201 Created` and the response body is a JSON object containing the output details.

```
POST /outputs
```

Request

Path	Type	Required	Description
streamId	string	yes	Stream identifier
format	**enum** string	yes	Output stream format [**HLS** | **DASH** | **RTMP** | **RTSP** | **WebRTC**]
sessionLimit	number	no	Number of concurrent sessions allowed
viewer	**Viewer**	no	Viewer details
callbackUri	string	no	Invokes on status change. (HTTP POST)

Table 16. Viewer

Path	Type	Required	Description
accessToken	string	no	Stream access token
ipAddress	string	no	IP address
geoLocation	**GeoLocation**	no	Geographical location

Example request

```
{
  "streamId": "731cbd85-7310-4e2e-8347-7396a30f9091",
  "format": "HLS"
}
```

Example request for "private" streams

```
{
  "streamId": "20b435ec-2ef7-4bba-b271-45cc03d7b1b6",
  "format": "WebRTC",
  "viewer": {
    "accessToken": "some_token"
  }
}
```

Response

Path	Type	Description
connection	**HttpConnection** | **WebRTCConnection**	Output endpoint parameters for viewer

Example response

```
{
  "connection": {
    "uri": "http://localhost:8080/hls/as+08189568-3492-4e15-92da-
f438eb56797d/index.m3u8"
  }
}
```

```
{
  "id": "6d26549d-ada3-4efb-b748-aaa8e3d2f45a",
  "viewer": {
    "accessToken": "some_token"
  },
  "connection": {
    "signallingUri": "wss://mydomain.com/webrtc/as+20b435ec-2ef7-4bba-b271-
45cc03d7b1b6"
  },
  "callbackUri": "http://mydomain.com/notifications"
}
```

Example callback notifications

```
{
  "streamId": "e011e662-b04a-4e8f-8b1b-1b9f7f685b2d",
  "viewerToken": "e200c009-d896-4427-ae46-78b5af240216",
  "statue": "STARTED"
}
```

```
{
  "streamId": "e011e662-b04a-4e8f-8b1b-1b9f7f685b2d",
  "viewerToken": "e200c009-d896-4427-ae46-78b5af240216",
  "statue": "STOPPED",
  "duration": 27,
  "upload": 20452500,
  "download": 42728
}
```

4.3.2. Retrieving outputs by stream id

You can also retrieve the contents of a outputs object by sending a GET request to the `/outputs` URL with query parameter "streamId".

```
GET /outputs
```

The response body is a JSON object containing list of outputs

Example response

```
[
{
  "id": "6d26549d-ada3-4efb-b748-aaa8e3d2f45a",
  "viewer": {
    "accessToken": "some_token"
  },
  "sessions": []
},{
  "id": "7256e563-e081-4efe-bc31-df44ea777f0c",
  "viewer": {
    "accessToken": "9287fb28-1635-4bd8-8c44-f09fe0945196"
  },
  "sessions": [
    "07bf40f7c5eb251f4ab9ab020ee983ce",
    "873a0c841b37066fbb49a906559592a7"
  ]
},{
  "id": "a959342d-c149-4b08-884e-16bf6a27b581",
  "viewer": {
    "accessToken": "4e6a9d93-fa21-429d-88bd-d0b0e19a3f01"
  },
  "sessions": []
}
]
```

4.3.3. Output sessions invalidation

You can invalidate all active sessions by sending a DELETE request to the `/outputs/:outputId/sessions` URL with output id.

```
DELETE /outputs/:outputId/sessions
```

4.3.4. Deleting output

To delete an output from the Ceeblue Cloud, send a DELETE request to `/outputs` with id. All sessions will be invalidated.

```
DELETE /outputs/:id
```

Chapter 5. Stream push

A “push” is a stream being sent to another host. It is a one-time only action, that is started and then disappears when the stream ends or the connection to the target is broken.

5.1. Add push

To create a new push send a POST request to the class URL containing the contents of the push output

```
POST /pushes
```

Request

Path	Type	Required	Description
format	**enum** string	yes	Push stream format [**RTMP** | **CMAF** | **UDPTS**]
streamId	string	yes	Stream identifier
output	**RTMPOutput** | **CMAFOutput** | **UDPTSOutput**	yes	Push endpoint details
autostart	boolean	no	Starts pushing on stream state **Ingestion**

Table 17. RTMPOutput

Path	Type	Required	Description
server	string	yes	RTMP server URL
key	string	no	RTMP stream key

Table 18. CMAFOutput

Path	Type	Required	Description
endpoint	string	yes	Streaming endpoint
unifiedStreamingPlatform	boolean	no	Enable unified streaming platform support

Table 19. UDPTSOutput (MPEG-TS over UDP)

Path	Type	Required	Description
ipAddress	string	yes	Streaming endpoint IP adress
port	number	yes	Streaming endpoint port
tracks	array of **TrackSelector**	no	List of track selectors (capture all if empty)

TrackSelector

Path	Type	Required	Description
type	const string	yes	Track type [**Video** | **Audio**]
idx	number	no	Track identifier
codec	const string	no	Codec type [**H264** | **VP8** | **VP9** | **AAC** | **MP3** | **Opus**]
resolution	**VideoResolutionSelector**	no	**Video** track selector by resolution

VideoResolutionSelector

Path	Type	Required	Description
width	number	yes	Picture width
height	number	yes	Picture height
approximate	boolean	no	Choose the track closest to the given pixel surface area

Example request

```
{
  "streamId": "4116f2d8-6c7c-402d-bd54-74ac3b8f06b4",
  "format": "RTMP",
  "output": {
    "server": "rtmp://live.myserver.com/push",
    "key": "news"
  }
}
```

```
{
  "streamId": "df3f3ae3-25de-495b-b221-4d01c734f1a5",
  "format": "CMAF",
  "output": {
    "endpoint": "http://ingest.xxx.yyy.someserver.com/live/4116f2d8-6c7c-402d-bd54-
74ac3b8f06b4/live.isml/stream",
    "unifiedStreamingPlatform": true
  },
  "autostart": true
}
```

```
{
  "streamId": "d11d2711-e64d-4885-81ae-13a87ce335b4",
  "format": "UDPTS",
  "output": {
    "ipAddress": "5.235.5.237",
    "port": 10210,
    "tracks": [
      {
        "type": "Video",
        "resolution": {
          "width": 1280,
          "height": 720,
          "approximate": true
        }
      },
      {
        "type": "Audio",
        "codec": "AAC"
      }
    ]
  }
}
```

Response

```
{
  "id": "df54ac18-a809-4d6e-9e64-a953c5161964",
  "streamId": "4116f2d8-6c7c-402d-bd54-74ac3b8f06b4",
  "format": "RTMP",
  "output": {
    "server": "rtmp://live.myserver.com/push",
    "key": "xxx"
  },
  "status": "Pending",
  "messages": []
}
```

`HTTP/1.1 201 Created` on success

5.2. Fetching stream push

A `GET` request fetches a specific stream push by id.

```
GET /pushes/:id
```

The response body is a JSON object

Example response

```
{
  "id": "e00b4183-a0d2-4340-b808-f3298e45bd72",
  "streamId": "4116f2d8-6c7c-402d-bd54-74ac3b8f06b4",
  "format": "RTMP",
  "output": {
    "server": "rtmp://live.myserver.com/push",
    "key": "xxx"
  },
  "status": "Pushing",
  "messages": []
}
```

5.3. Deleting stream

To delete an stream push from the Ceeblue Cloud, send a DELETE request to `/pushes` with id.

```
DELETE /pushes/:id
```

Response
`HTTP/1.1 204 No Content` on success

5.4. Retrieving pushes by stream id

You can also retrieve the push ids of a stream object by sending a GET request to the `/pushes/stream/[id]` URL.

```
GET /pushes/stream/:id
```

The response body is a JSON object containing list of push ids

Example response

```
[
  "3cc92681-6515-423b-bb9e-46a3bd13c98e",
  "4ddac41c-b45b-473f-948e-19be17417df0",
  "706d9df8-8c81-4be4-b789-117d8c1c249c",
  "8a3a5f8c-1d5b-4237-845e-9aa1d11c0f0c",
  "a6786b13-46f4-4b00-8fa2-09dc86751ce0",
  "c5ec735b-026d-4931-9b7f-158f869fb1ef",
  "cf5eccb7-0a32-4275-a420-42d8836b73e4",
  "df54ac18-a809-4d6e-9e64-a953c5161964",
  "f63d6b9e-a6da-478e-9430-d182f5a7c8e4"
]
```

Chapter 6. Stream recording

6.1. Add recording

To create a new recording send a `POST` request to the class URL containing the contents of the recording

```
POST /recordings
```

Recording

Path	Type	Required	Description
streamId	string	yes	Stream identifier (**Stream**)
fileName	string	yes	Target file name
fileFormat	const string	yes	Target file format [**MKV**]
capture	**Capture**	yes	Stream capture settings
storageId	string	yes	Storage identifier (**Storage**)
callbackUri	string	no	Invokes on state change. (HTTP POST)

Capture

Path	Type	Required	Description
source	const string	yes	Capturing source [**Incoming** | **Outgoing**]
tracks	array of **TrackSelector**	yes	List of track selectors (capture all if empty)

TrackSelector

Path	Type	Required	Description
type	const string	yes	Track type [**Video** | **Audio**]
idx	number	no	Track identifier

Example request (capture all outgoing tracks)

```
{
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": []
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "callbackUri": null
}
```

Example request (capture some outgoing tracks)

```
{
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": [
      { "type": "Audio", "idx": 0 },
      { "type": "Video", "idx": 3 },
      { "type": "Video", "idx": 2 },
      { "type": "Video", "idx": 5 }
    ]
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "callbackUri": null
}
```

Response

When the creation is successful, the HTTP response is a `201 Created` and the response body is a JSON object containing the recording id, etc.

Path	Type	Description
id	string	Recording identifier
streamId	string	Stream identifier
fileName	string	Target file name
fileFormat	string	Target file format [**MKV**]
capture	object	Capturing settings (**Capture**)
storageId	string	Storage identifier
state	const string	Recording state [**Recording** | **Uploading** | **Completed** | **Error**]

Path	Type	Description
duration	number	Recording duration (seconds)
length	number	Recording file length (bytes)
uploaded	number	Amount of recorded file uploaded to storage (bytes)
messages	array of string	List of messages

Example response (capture some outgoing tracks)

```
{
  "id": "19a475cb-ae62-40c0-ad1c-e1b9a51e713e",
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": []
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "state": "Recording",
  "duration": "0",
  "length": "0",
  "uploaded": "0",
  "messages": [],
  "callbackUri": null
}
```

Example response (capture all outgoing tracks)

```
{
  "id": "50024760-936d-4131-a9d6-2ebd74f754a9",
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": [
      { "type": "Audio", "idx": 0 },
      { "type": "Video", "idx": 3 },
      { "type": "Video", "idx": 2 },
      { "type": "Video", "idx": 5 }
    ]
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "state": "Recording",
  "duration": "0",
  "length": "0",
  "uploaded": "0",
  "messages": [],
  "callbackUri": null
}
```

6.2. Fetching recording

A `GET` request fetches a specific recording by id.

```
GET /recordings/:id
```

The response body is a JSON object

Example completed response

```
{
  "id": "19a475cb-ae62-40c0-ad1c-e1b9a51e713e",
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": []
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "state": "Completed",
  "duration": "985",
  "length": "1142426340",
  "uploaded": "1142426340",
  "messages": [],
  "callbackUri": null
}
```

6.3. Fetching recordings

```
GET /recordings
```

6.4. Fetching recordings by stream id

```
GET /recordings/stream/:id
```

The response body is a JSON object containing list of recordings

Example response

```
[{
  "id": "0a719430-18fc-4108-a4da-97e4864a8d13",
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample 1",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": []
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "state": "Uploading",
  "duration": "385",
  "length": "430507022",
  "uploaded": "0",
  "messages":[],
  "callbackUri":null
},{
  "id": "1c7c52d9-cb9d-4fd2-a8de-1c9aeed4422f",
  "streamId": "a347d841-f4a5-46bf-ab13-8d77435d01e6",
  "fileName": "Sample 2",
  "fileFormat": "MKV",
  "capture": {
    "source": "Outgoing",
    "tracks": []
  },
  "storageId": "aca7d52e-2b3f-4776-9060-85e656d00f79",
  "state": "Recording",
  "duration": "375",
  "length": "419369924",
  "uploaded": "0",
  "messages": [],
  "callbackUri": null
  }
]
```

6.5. Stop recording

To stop recording send a `PUT` request by recording id

```
PUT /recordings/:id/stop
```

6.6. Deleting recording

To delete a recording from the Ceeblue Cloud, send a `DELETE` request to `/recordings` with id.

```
DELETE /recordings/:id
```

Response

`HTTP/1.1 204 No Content` on success.

Chapter 7. Stream snapshot

7.1. Fetching stream snapshot

A `GET` request fetches a specific stream snapshot by stream id and capture source.

```
GET /snapshots/stream/:id/:source/image
```

- **id** - stream identifier
- **source** - capturing source [**incoming** | **outgoing**]

The response body is a binary object with content type `'image/png'` or `'image/jpg'`

7.2. Setup snapshot settings

To setup the creation of snapshots send a `PUT` request to the class URL containing the contents of the setting

```
PUT /snapshots/stream/:id/:source
```

- **id** - stream identifier
- **source** - capturing source [**incoming** | **outgoing**]

Recording

Path	Type	Required	Description
format	const string	yes	Target file format [**PNG** | **JPEG**]
quality	const string	yes	Target file quality [**Low** | **Medium** | **High**]
interval	number	yes	Snapshot producing interval [**1** - **30**]

Example request (JPEG snapshots every 5 seconds)

```
{
  "format": "JPEG",
  "quality": "Medium",
  "interval": 5
}
```

Response
When the setup is successful, the HTTP response is a `200 Created`

7.3. Fetching snapshot settings

A `GET` request fetches a specific snapshot settings

```
GET /snapshots/stream/:id/:source
```

- **id** - stream identifier
- **source** - capturing source [**incoming** | **outgoing**]

The response body is a JSON object

Example response

```
{
  "format": "JPEG",
  "quality": "Medium",
  "interval": 5
}
```

7.4. Clear snapshot settings

To clear a snapshot settings, send a `DELETE` request

```
DELETE /snapshots/stream/:id/:source
```

- **id** - stream identifier
- **source** - capturing source [**incoming** | **outgoing**]

Response
`HTTP/1.1 204 No Content` on success.

Chapter 8. Stream information

8.1. Fetching stream information

A `GET` request fetches a specific stream information by stream id and capture source.

```
GET /info/stream/:id/:source
```

- **id** - stream identifier
- **source** - capturing source [**incoming** | **outgoing**]

The response body is a JSON object containing a list of stream tracks with details

Example response

```
[
  {
    "idx": 2,
    "type": "Video",
    "codec": "H264",
    "bps": 499364,
    "width": 1920,
    "height": 1080
  },
  {
    "idx": 3,
    "type": "Video",
    "codec": "H264",
    "bps": 249331,
    "width": 1280,
    "height": 720
  },
  {
    "idx": 4,
    "type": "Video",
    "codec": "H264",
    "bps": 186876,
    "width": 960,
    "height": 540
  },
  {
    "idx": 5,
    "type": "Video",
    "codec": "H264",
    "bps": 123957,
    "width": 640,
    "height": 360
  },
```

```
  {
    "idx": 6,
    "type": "Video",
    "codec": "H264",
    "bps": 61740,
    "width": 256,
    "height": 144
  },
  {
    "idx": 0,
    "type": "Audio",
    "codec": "AAC",
    "bps": 15998,
    "channels": 2,
    "rate": 48000
  },
  {
    "idx": 1,
    "type": "Audio",
    "codec": "opus",
    "bps": 16000,
    "channels": 2,
    "rate": 48000
  }
]
```

Chapter 9. Storage

9.1. Add storage

To create a new storage send a `POST` request to the class URL containing the contents of the storage

```
POST /storages
```

Storage

Path	Type	Required	Description
name	string	yes	Storage name
type	**enum** string	yes	Storage type [**AmazonS3** | **AmazonS3Compatible**]

AmazonS3 or AmazonS3Compatible data parameters.

Path	Type	Required	Description
accessKeyId	string	yes	An access key ID (for example, AKIAIOSFODNN7EXAMPLE)
secretAccessKey	string	yes	A secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)
bucket	string	yes	Destination bucket
path	string	no	Destination path inside bucket

AmazonS3Compatible data parameters.

Path	Type	Required	Description
endpoint	string	yes	A URL that is the entry point for a web service

Example AmazonS3 request

```
{
  "name": "Demo storage",
  "type": "AmazonS3",
  "accessKeyId": "AK...............PU",
  "secretAccessKey": "GW.............................f+",
  "bucket": "demo-recordings",
  "path": null
}
```

Example AmazonS3 response

```
{
  "id":"868434cf-a8c0-4ded-a17d-469fadc593b4",
  "type": "AmazonS3",
  "accessKeyId": "AK...............PU",
  "secretAccessKey": "GW.............................f+",
  "bucket": "demo-recordings",
  "path": null
}
```

Example AmazonS3Compatible request

```
{
  "name": "Demo storage",
  "type": "AmazonS3Compatible",
  "accessKeyId": "AK...............PU",
  "secretAccessKey": "GW.............................f+",
  "bucket": "demo-recordings",
  "path": null,
  "endpoint": "cloud.storage.com"
}
```

Example AmazonS3Compatible response

```
{
  "id":"868434cf-a8c0-4ded-a17d-469fadc593b4",
  "type": "AmazonS3Compatible",
  "accessKeyId": "AK...............PU",
  "secretAccessKey": "GW.............................f+",
  "bucket": "demo-recordings",
  "path": null,
  "endpoint": "cloud.storage.com"
}
```

9.2. Fetching storage

A `GET` request fetches a specific storage by id.

```
GET /storages/:id
```

The response body is a JSON object.

Example response

```
{
  "id":"868434cf-a8c0-4ded-a17d-469fadc593b4",
  "type": "AmazonS3",
  "accessKeyId": "AK...............PU",
  "secretAccessKey": "GW.............................f+",
  "bucket": "demo-recordings",
  "path": null,
  "endpoint": "cloud.storage.com"
}
```

9.3. Retrieving storages

```
GET /storages
```

The response body is a JSON object containing list of storages

Example response

```
[
  {
    "id": "2c8b5135-f603-453f-8bac-33cae9ac1a97",
    "name": "Compatible demo storage",
    "type": "AmazonS3Compatible",
    "accessKeyId": "E..................E",
    "secretAccessKey": "xB....................................jS",
    "bucket": "recordings-demo",
    "path": null,
    "endpoint": "cloud.storages.com"
  },{
    "id": "868434cf-a8c0-4ded-a17d-469fadc593b4",
    "name": "Demo storage",
    "type": "AmazonS3",
    "accessKeyId": "AK................PU",
    "secretAccessKey": "GW...................................rf+",
    "bucket": "recordings-demo",
    "path": null
  }
]
```

9.4. Update storage

A `PUT` request updates storage by id.

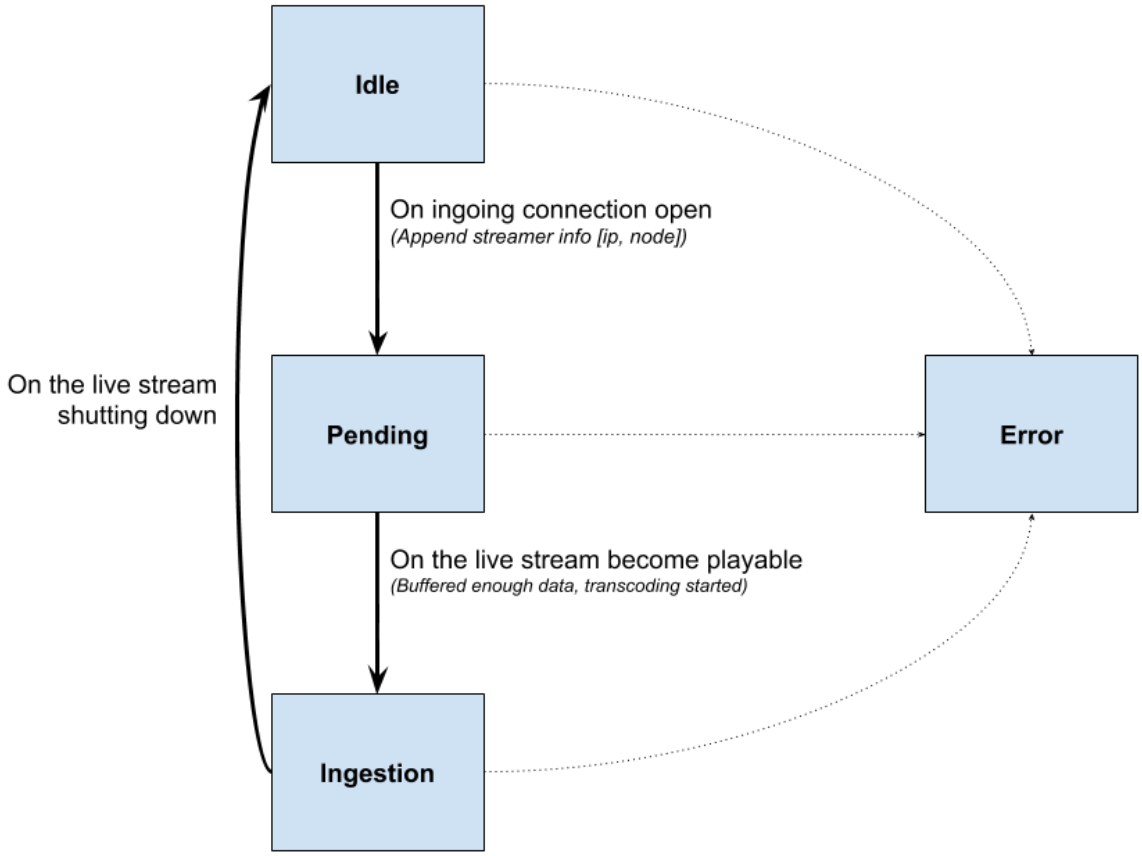
```
PUT /storages/:id
```

See **Storage**

9.5. Delete storage

To delete a storage from the Ceeblue Cloud, send a `DELETE` request to `/storages` with id.

```
DELETE /storages/:id
```

Response
`HTTP/1.1 204 No Content` on success.

Appendix A: Stream status