Task-Engine Documentation

VUALTO

Dec 19, 2018

Contents

CHAPTER 1

Contents:

1.1 TASK ENGINE API

1.1.1 Introduction to Task Engine

Task Engine is a product that facilitates easy integration between client systems and the vudrm platform. It is designed to allow simple or complicated workflows to be constructed and tested easily and quickly. Task Engine is exposed via a simple API and reports back to the client system via any required mechanism, although usually this is as simple as an HTTP POST to an endpoint.

How it works

Task Engine breaks a workflow up in to several component parts. Every piece of work submitted is a job, and each job is broken up in to a series of tasks. These tasks are then scheduled on to queues which workers then process.

Every job and task have unique identities, and these are exposed back to the client system both through the response to the initial job submission and through the callbacks (unless for some reason the configuration prevents this from happening).

When a task is being processed, a callback is sent through the configured mechanism as it starts and when it completes or fails, this way the client system should be able to track the progress of a job as it progresses through the task engine and then perform any relevant tasks when the job as a whole is completed.

System Requirements

The Task Engine is designed to run in Docker containers, and is therefore able to run on any Docker-supported Linux host, preferably running whatever the latest version of Docker is at the time (currently 1.12.1). Any workers that need to run on Windows are currently NOT containerized but are installed as a native Windows service – this will change as Docker support for Windows matures.

When a worker starts a task, it will generally need to copy one or more files to local storage to perform its work in an efficient manner, so it's critical that hosts have sufficient disk space for processing to take place; this is particularly

important when dealing with video transcoding or DRM encryption. This also means that sometimes running more than one worker per host is not an option. Vualto will usually advise on disk space requirements once the workflow is fully defined and some representative test content has been processed.

1.1.2 Integration

There are two main integration points for the task engine:

Triggers – how you notify task engine that a workflow needs to be executed, usually initiated by a call to the Task Engine API.

Callbacks – how we notify client systems of job progress.

1.1.3 Triggers

While the task engine can be configured to use watch folders or other methods of notification (currently outside the scope of this document) to trigger a workflow, the most common route is to have the client system trigger a workflow via an API call.

There is a single endpoint, /job, that handles all requests, only the payload varies according to the requirements of the workflow. As most clients have varying workflows, this document will give an examples of some of the most common workflows.

Custom workflows are supported, however; these would require additional development. The main differences will be the name of the workflow and the parameters passed in the 'parameters' section.

Workflow Trigger Example

An example call to the the Task Engine API to trigger a workflow would be as follows (Parameters must be a parameters object):

```
curl -X POST \
-H "Content-Type: application/json" \
-H "API-KEY: <client_api_key>" \
-d '{"client": "<client_name>",
"job": { "workflow": "<workflow_name>" },
"parameters": {
    "content_id": "<unique_content_id>",
    .
    .
    .
    }}' "http://<vuflow_server>/job"
```

The Headers are defined as follows:

More information about specific workflows can be found here.

1.1.4 Rest Endpoint Callbacks

Specifying the Task Engine `https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback` "webhook" in the callback will register the VOD asset within the Vualto vuflow events management system. It's recommended and this is optional but does mean you can view the VOD event within `https://admin.vuworkflow.staging.vualto.com`. This is also required for VOD assets to automatically go into private mode (configurable) after ingestion.

The callbacks will send JSON like below. Please note that `status` can come back as `failed` or `completed`. You will also be concerned with the **message**. This will be the resulting filename after creating the VOD stream or switching DRM. For example, you will require this for building your streaming Urls. For DASH it would look something like:

`.../054b58a1-8f05-4f08-bb17-b4698e3e9d1d_drm_57d82537-1775-4e29-97e4-bba86e0219fb.ism/.mpd`

Retrieving Urls is also possible within the VIS API if you don't want to build and generate these Urls yourself. All callbacks will send the following header and value. This can be used to validate the request:

`Workflow-Id:  52ec321b-8da3-4f5f-9cd3-749789577cd3`

Before and after each step in a workflow, Task Engine can provide a callback to a given endpoint. Given that each workflow is different, the specific task names will vary, however the callbacks will generally be made as follows:

```
curl -X POST \
-H "Content-Type: application/json" \
-H "<any_required_header>: <any_required_value>" \
... \
-d '{"job_id": <job_id>,
    "task_id": <task_id>,
    "task_name": "<name>",
    "content_id": "<content_id>",
    "event": "<start | complete | fail>",
    "message": "<message>",
    "datetime": "yyyy-mm-ddThh:mm:ssZ"}' \
    "<configured_callback_endpoint>"
```

As Task Engine is an integration product that can be customised, any specific requirements are easily catered for (i.e. specific headers being set for authentication, notification via SNS, RabbitMQ, email etc), although the data available to be sent will remain the same as listed above.

1.2 TASK ENGINE WORKFLOWS

1.2.1 Vodstream

This workflow will create a server side manifest, with and/or without DRM, that can be used for on the fly delivery of VOD content via USP.

Vodstream: Parameters

Vodstream: JSON Payload example

```
{
  "client": "staging",
  "job": {
    "workflow": "vodstream"
  },
  "parameters": {
    "content_id": "demo1",
    "source_folder": "mz-ast-2055fcff-8cca-4e37-85b9-9647dbe50398-1",
    "delete_source": false,
    "encrypted": true,
```

(continues on next page)

(continued from previous page)

```
    "output_folder": "mz-ast-2055fcff-8cca-4e37-85b9-9647dbe50398-1",
    "drm": [
      "fairplay",
      "playready",
      "widevine"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://aaa.com/end",
      "http://bbb.com/end"
    ],
    "create_thumbnail": true,
    "thumbnail_time": "1:34.000",
    "generate_mp4": true,
    "mp4_filename": "demo_sample.mp4",
    "combine_sources": true,
    "create_dref": true,
    "all_audio_tracks": true
  }
}
```

Vodstream: Callback properties

Task Callback:

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the defualt properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

1.2.2 Vodcapture

This workflow allows you to create a frame accurate vod clip by passing in a start and end UTC time stamp. The result will be an MP4 on disk.

Vodcapture: Parameters

Vodcapture: JSON Payload example

```
{
  "client": "staging",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:00:00.000",
```

(continues on next page)

(continued from previous page)

```
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    "encrypted": false,
    "drm": [
        "fairplay",
        "playready",
        "cenc",
        "widevine",
        "aes"
    ],
    "frame_accurate": true,
    "copy_ts": false,
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ],
    "key_id": "346AS5847333DDSHKFSDS7633429CD33",
    "content_key": "346AS5847333DDSHKFSDS7633429CD33",
    "output_file": "demo_sample.ismv",
    "create_thumbnail": true,
    "thumbnail_time": "1:34.000",
    "generate_mp4": true,
    "mp4_filename": "demo_sample.mp4",
    "create_dref": true
  }
}
```

Vodcapture: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the defualt properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

1.2.3 Voddeletes3

This workflow allows you to delete content on S3.

Voddeletes3: Parameters

Voddeletes3: JSON Payload example

```
{
  "client": "staging",
  "job": {
    "workflow": "voddeletes3"
  },
```

(continues on next page)

(continued from previous page)

```
  "parameters": {
    "content_id": "demo1",
    "folder": "vualto-test-1",
    "rest_endpoints": [
            "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/
↪taskenginecallback",
            "http://your.custom.endpoint"
        ]
  }
}
```

Voddeletes3: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the defualt properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

1.2.4 Drmswitch

This workflow allows you to toggle DRM on and off.

Drmswitch: Parameters

Drmswitch: Payload example

```
{
  "client": "staging",
  "job": {
    "workflow": "drmswitch"
  },
  "parameters": {
    "content_id": "demo1",
    "folder": "vualto-test-1",
            "rest_endpoints": [
            "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/
↪taskenginecallback",
            "http://your.custom.endpoint"
        ]
  }
}
```

Drmswitch: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the defualt properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

1.2.5 CreateMP4

This workflow allows you to create an MP4 from a VOD asset

CreateMP4: Parameters

CreateMP4: Payload example

```
{
  "client": "staging",
  "job": {
    "workflow": "createmp4"
  },
  "parameters": {
    "content_id": "demo1",
    "folder": "vualto-test-1",
            "rest_endpoints": [
            "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/
↪taskenginecallback",
            "http://your.custom.endpoint"
    ],
    "mp4_filename": "result.mp4",
    "output_folder": "vualto-test-1/downloads"
  }
}
```

CreateMP4: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the defualt properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

1.2.6 Additional Workflow Features

Priority

The Task Engine supports ordering of jobs by priority. The priority parameter can be submitted as part of the json payload being submitted. The priority is in ascending order as follows:

1 - Top Priority..5 - Default..10 - Least Priority

The `"priority"` parameter needs to be submitted within the `"job"` section of the json payload as shown below:

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture",
    "priority": 3
  },
  "parameters": {
    "content_id": "demo1",
    ...
    ...
    ...
  }
}
```

Whenever an execution slot is available, the system will first check by priority and then check the submission time and date of the job. In the case where multiple jobs are executed with the same priority (eg. with the default priority 5), the Task Engine operates in a FIFO (First In First Out) manner.

Multiple Clips

The Task Engine includes a feature that will allow multiple clips to be stitched togther into a single clip, in a single job. This can be done by defining multiple objects within the `"clips"` parameter in the json payload for `vodcapture`. This also allows a mixture of live and VoD sources to be captured and stitched toghether into a new clip. The example below shows how the `"clips"` parameter would need to be provided to achive this.

```
{
  "client": "staging",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/copyright.ism/manifest"
      },
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      },
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:35:00.000",
        "end": "2018-06-06T11:00:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    ...
    ...
    ...
  }
}
```

Multiple Sources

In some cases, a live stream could have multiple origins setup (eg. for load balancing the origin servers). The Task Engine, allows for both streams to be defined as the source for a capture. It is smart enough to find which live stream will provide the best output capture and use that stream as the source. If the Task Engine discovers discontinuities within the streams, it will use segments from both streams to try and generate a clip with the least number of missing fragements.

The streams can be defined in the `"sources"` parameter when executing the `vodcapture` workflow.

```
{
  "client": "staging",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "sources": [
          "http://mydomain.com/live_1.isml/manifest",
          "http://mydomain.com/live_2.isml/manifest"
        ],
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    ...
    ...
    ...
  }
}
```

In this case, `"sources"` replaces the `"source"` parameter, however; it can still be used in conjunction with other clips which only contain a single stream as shown below.

```
{
  "client": "staging",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/copyright.ism/manifest",
      },
      {
        "sources": [
          "http://mydomain.com/live_1.isml/manifest",
          "http://mydomain.com/live_2.isml/manifest"
        ],
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
```

(continues on next page)

(continued from previous page)

```
      }
    ],
    ...
    ...
    ...
  }
}
```

Generate Download Clips

The Task Engine `vodcapture` workflow supports generating download clips without creating VoD assets. This is done by setting the property `"generate_vod"` to false and `"generate_mp4"` to true. It is important that if `"generate_vod"` is set to false, to not manually override the `"create_dref"` parameter. Setting `"create_dref"` to true will lead to a failed workflow as this requires VoD assets to generate DREF mp4s.

The resulting downlaod will be an MP4 containg all the video, audio and caption tracks defined using the clip's `"filter"` parameter. If no filter is defined, the resulting MP4 will contain all the tracks availble in the stream.

Scheduler

The Task Engine supports scheduling of jobs via a `run_at` attribute. Jobs are moved from a queue_state of `scheduled` to a queue_state of `queued` via a scheduler-worker. The interval at which this runs is pulled from the database settings table (schedule_interval, default: 1 hour).

The scheduler looks for jobs which have a queue_state of `scheduled` and a `run_at` time in the past

The schedule_interval can be set via an api call. (where x is time in seconds)

`post '/settings'`

```
{
  "client": "demo-client",
  "name": "schedule_interval",
  "setting": "<x>"
}
```

A jobs `run_at` attribute can be set in multiple ways and defaults to the time it was created at.

1. When submitting a job

`post /job/:job_id`

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture",
    "run_at": "1970-01-01T00:00:00"
  }
```

1. When updating an existing job

`put /jobs/:job_id`

```
{
  "client": "demo-client",
```

(continues on next page)

(continued from previous page)

```
  "run_at": "1970-01-01T00:00:00"
}
```

1. When submitting a capture with a clip end time in the future

if a capture is submitted with a clip end time that is in the future, it will be automatically scheduled to run at the end time of the clip which is furthest in the future. Unless the `"run_at"` time (if specified) is further in the future than the end time.

1.2.7 Workflow Trigger Example

Example of a curl command to trigger ingest for the vodstream workflow:

```
curl -X POST \
http://vualto.demo.com/job \
-H "API-KEY: aabbccdd-1122-3344-5566-eeff77889900" \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' \
-d '{
  "client": "vualto",
  "job": {
    "workflow": "vodstream"
  },
  "parameters": {
      "content_id": "demo_1",
      "source_folder": "/input/demo1",
      "delete_source": false,
      "encrypted": false,
      "output_folder": "/test",
      "drm": [
        "fairplay",
        "playready",
        "cenc",
        "widevine"
      ],"rest_endpoints": [
        "https://webhook.site/55151d14-cee1-416b-b956-a90525ae8f58",
        "https://webhook.site/bc4c13ee-f118-4d5b-a4af-7ac07890a7f1"
      ],
      "create_thumbnail": false,
      "generate_mp4": true,
      "combine_sources": true,
      "create_dref": true,
      "all_audio_tracks": false,
    }
  }
```

This results in the files `<content_id>_<drm_tag>_<unique_guid>.ism` and `<content_id>_<unique_guid>.ismv` being produced in the folder:

`<configured_root>(/<optional_output_folder>)/<content_id>`

The response from this call should be either a 200 OK, with the following payload:

`{ "job_id": <job_id>, "result": "accepted" }`

or a 400 BAD REQUEST with the following payload:

`{ "error": "<description_of_error>" }`

Assuming the call is successful, this would add an ingest job to the Task Engine queue, with the files to be ingested expected to be in the following location:

`<input_root>/input/demo1`

If the process completes successfully, then the output would be the following files:

`<output_root>/test/demo1.ism` `<output_root>/test/demo1.ismv`

If the 'output_folder' parameter was excluded, then the files would be output to the following locations:

`<output_root>/demo1.ism` `<output_root>/demo1.ismv`

NOTE: there may be some additional files, depending on the exact processes involved, but the minimum would usually be these.