

# 使用GCP Transcoder API构建自动化转码 workflow

Author: [eugeneyu@google.com](mailto:eugeneyu@google.com)

## [概述](#)

[谷歌云转码服务简介](#)

[创建转码模板Job Template](#)

[提交转码任务](#)

[创建PubSub消息队列接收转码完成通知](#)

[使用Cloud Functions自动触发转码任务](#)

[使用Cloud Functions自动发布转码后视频](#)

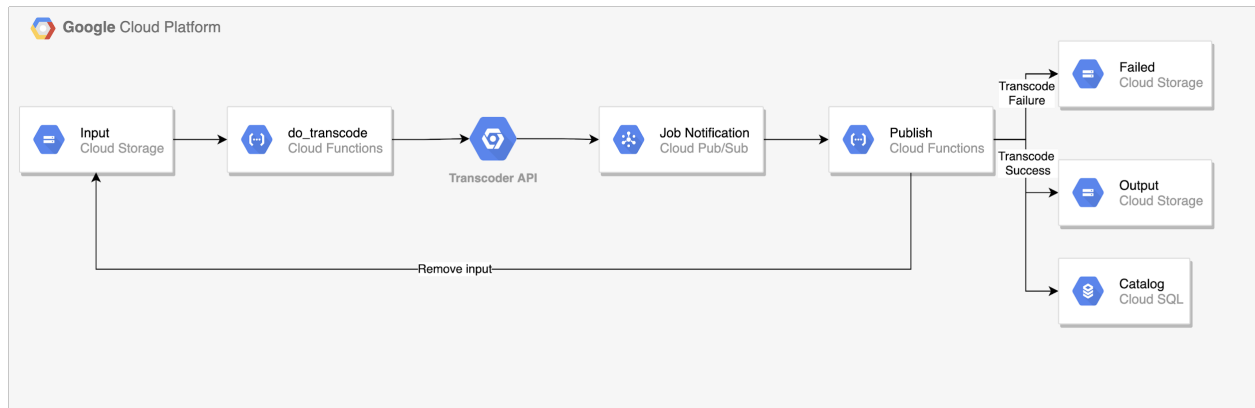
## [附录](#)

[A. 参考文档和工具](#)

[B. Job Template示例](#)

## 概述

本文介绍谷歌云Transcoder API的基本使用，以及应用Transcoder API, Cloud Functions, Cloud Pub/Sub等服务搭建自动化视频转码 workflow 的通用方案。该方案的架构如下图所示。后面的章节会介绍一些关键步骤的实现方法和示例。



以下为配置自动化转码任务的主要步骤。

1. 在本地配置谷歌云命令运行环境
2. 在谷歌云对象存储服务创建输入存储桶和输出存储桶
3. 创建转码Job Template，做自定义转码配置
4. 上传待转码视频文件并提交转码任务
5. 轮询转码状态，或者使用消息队列接收转码完成通知

因为目前谷歌云转码服务暂时还不支持控制台界面话配置，下面的步骤都是使用REST API完成。为了生成鉴权的OAuth Token，也需要执行命令的环境安装谷歌云gcloud命令行工具，具体参考[4] [Installing Google Cloud SDK](#)。也可以在控制台Cloud Shell内执行命令，该环境自带谷歌云命令行工具。

## 谷歌云转码服务简介

谷歌云Transcoder API目前可以支持如下功能。

- **不同容器格式的输出**，包括 MPEG-4 (MP4)、基于 HTTP 的动态自适应流式传输 ( DASH ，也称为 MPEG-DASH ) 和 HTTP Live Streaming (HLS)
- 以不同的比特率和分辨率输出
- 以编程方式优化视频输出，包括：
  - **剪裁视频尺寸**
  - 插入**重叠式图片或动画**
- 配置低层级编码参数，例如比特率
- 使用综合的编辑列表重新合成现有媒体内容
- 通过数字版权管理 (DRM) 系统加密内容以保护内容，包括：
  - 适用于 HLS 格式的 FairPlay 流式传输
  - 适用于 MPEG-DASH 格式的 PlayReady
  - 适用于 Chromium Web 浏览器和 Android 上的 MPEG-DASH 和 HLS 的Widevine
- 插入广告关键帧以允许视频播放器客户端插入广告

- 使用生成的[视频帧精灵表](#)创建缩略图。
- 创建作业模板以保存和重复使用自定义或复杂的配置，从而对作业进行转码

对于转码的媒体文件，支持如下输入编码格式

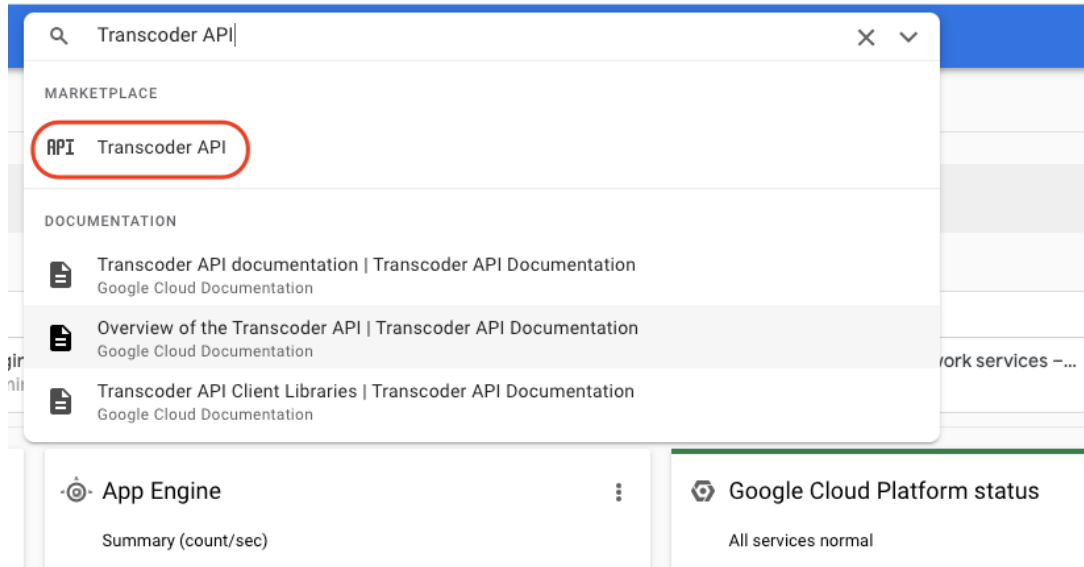
容器	AVI、GXF、MKV (Matroska)、MOV、MPEG2-TS、MP4、MXF (仅 OP1A) 、WMV
视频编解码器	DNxHD、DV/DVProHD、DV25、DV50、H.261、H.262、H.263、H.264 (AVC)、H.265 (HEVC)、MPEG-1、MPEG-2、MPEG-4 第 2 部分、ProRes、Theora、VC-1、VP8、VP9、XDCAM
音频编解码器	AAC、AC3、AIFF、E-AC3、MP3、Opus、PCM、WAV、WMA、WMA2、Vorbis
字幕	SCC、SRT、VTT
叠加层	JPG

支持如下输出编码格式

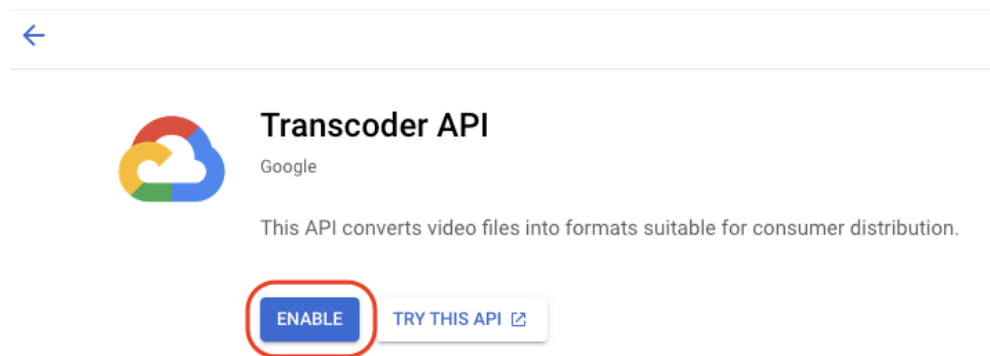
容器	Apple HLS (使用 MPEG2-TS、fMP4) 、MPEG-DASH (使用 fMP4) 、MP4
视频编解码器	H.264 (AVC)、H.265 (HEVC)、VP9
音频编解码器	AAC-HE、AAC-HEv2、AAC-LC、AC3、E-AC3、MP3
字幕	CEA-608/708、WebVTT
映像	JPG 图块、单个图片

## 开通Transcoder API

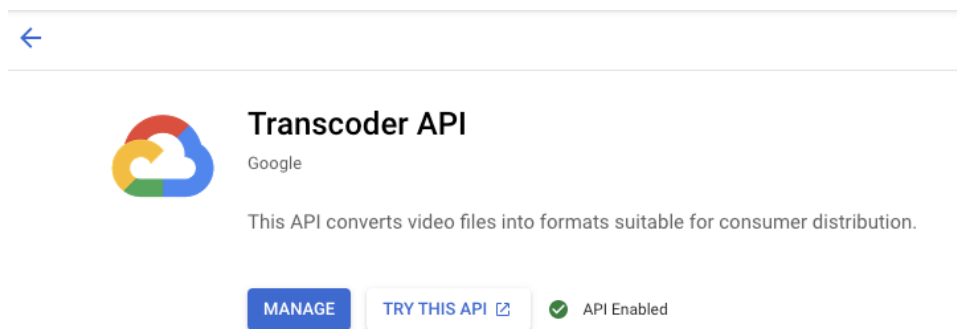
可以用下面步骤在谷歌云控制台开通Transcoder API。首先在控制台搜索栏搜索Transcoder API，并进入Transcoder API管理界面。



在Transcoder API管理界面，如果该服务没有开通，会有一个蓝色ENABLE按钮。点击此按钮以开通此服务。开通服务并不会产生任何费用。



服务开通后，可以点击MANAGE按钮来查看服务使用状态。



# 创建转码模板Job Template

根据业务的需要，参照文档[3] [JobConfig reference](#)来创建Job Template（转码模板）。下面为将输入文件转码为多码率H264编码HLS视频流的模板样例。

## [附录B. Job Template示例](#)

该示例也可以从Github地址下载。

<https://github.com/eugeneyu/cloud-demos/blob/master/transcode/transcode-template-multi-bitrate.json>

要创建Job Template，首先配置环境变量

```
PROJECT_ID=youzhi-lab
LOCATION=asia-east1
TEMPLATE_ID=multi-bitrate-20200106
```

其中PROJECT\_ID是谷歌云项目的ID，LOCATION是使用转码服务的谷歌云区域，TEMPLATE\_ID是客户自己定义的转码模板ID。

然后执行下面命令来创建Job Template转码模板。

```
curl -X POST \
-H "Authorization: Bearer "$(gcloud auth application-default
print-access-token) \
-H "Content-Type: application/json; charset=utf-8" \
-d @transcode-template-multi-bitrate.json \
https://transcoder.googleapis.com/v1beta1/projects/$PROJECT_ID/locations/$LOCATION/jobTemplates?jobTemplateId=$TEMPLATE_ID
```

注意，Job Template不能更新，只能新加，或者对现有的删除后重建。删除Job Template可以用下面命令。

```
curl -X DELETE \
-H "Authorization: Bearer "$(gcloud auth application-default
print-access-token) \
https://transcoder.googleapis.com/v1beta1/projects/$PROJECT_ID/locations/$LOCATION/jobTemplates/$TEMPLATE_ID
```

转码配置还要注意下面几点。

- 如果使用Fragmented MP4 (fmp4) 封装格式, 可以用同一组媒体文件来提供DASH和HLS流, 仅是播放列表内容不一样, 这样可以节省存储, 方便管理。但是要注意, fmp4封装不能混装(multiplex)音频和视频流在一个文件里, 而需要把音频单独输出成一个文件。
- 如果使用传统的TS封装格式来提供HLS流, 要注意可以使用H264或H265视频编码, 但是不能使用VP9编码。

## 提交转码任务

转码任务配置主要指定输入文件, 输出路径, 和转码模板。输入文件需要是在谷歌云存储中的媒体文件。

```
BUCKET_NAME=hk-publish
GCS_INPUT_VIDEO=video/robocar.mp4
GCS_OUTPUT_FOLDER=transcode_output/robocar_2020010601/

curl -X POST
"https://transcoder.googleapis.com/v1beta1/projects/${PROJECT_ID}/locations/${LOCATION}/jobs" \
-H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
-H "Content-Type: application/json" \
-d \
'{
  "inputUri": "gs://${BUCKET_NAME?}/${GCS_INPUT_VIDEO?}",
  "outputUri": "gs://${BUCKET_NAME?}/${GCS_OUTPUT_FOLDER?}",
  "templateId": "'${TEMPLATE_ID?}'"
}'
```

如果提交任务顺利, 命令行会输出任务的信息, 其中含有如下的任务名称。

```
{
  "name":
  "projects/247839977271/locations/asia-east1/jobs/6020138d0faf9878e2079bb3fb35a6e3",
  "config": {
    .....
  }
}
```

可以用命令轮询任务的执行情况，直到任务完成。

```
JOB_NAME=projects/247839977271/locations/asia-east1/jobs/7b062305381b90dc9173b8e2eba94e02

curl https://transcoder.googleapis.com/v1beta1/$JOB_NAME \
-H "Authorization: Bearer "$(gcloud auth application-default
print-access-token)
```

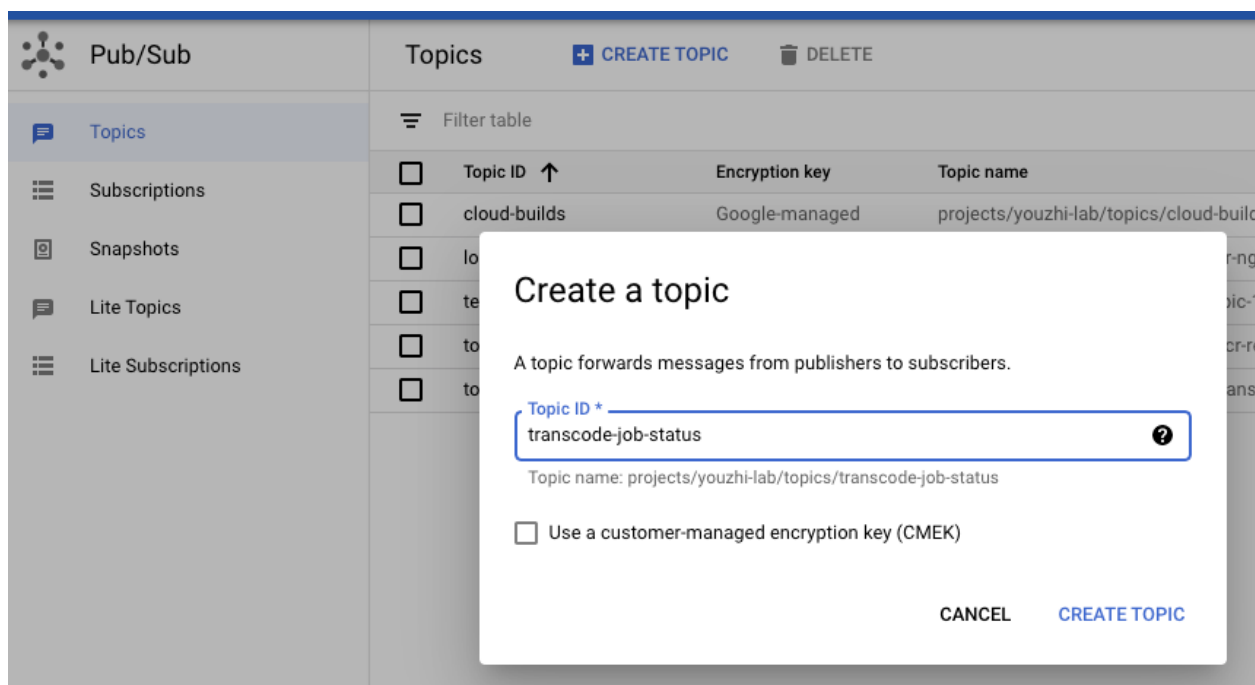
查询任务信息可以看到类似下面的输出。

```
}
  },
  "state": "SUCCEEDED",
  "createTime": "2021-01-06T05:21:38.813Z",
  "startTime": "2021-01-06T05:21:46.284Z",
  "endTime": "2021-01-06T05:37:45.728Z",
  "ttlAfterCompletionDays": 30
}
```

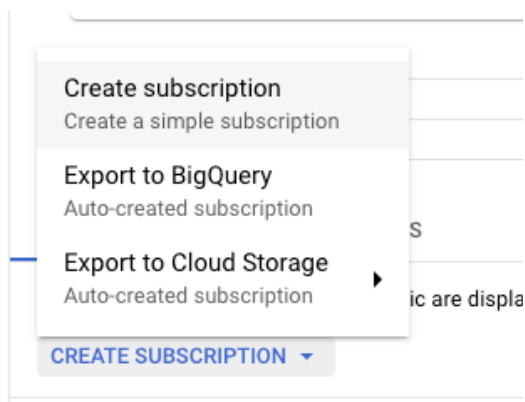
当state为RUNNING时，任务仍在进行。当其为SUCCEEDED时，表示任务成功完成。

## 创建Pub/Sub消息队列接收转码完成通知

可以配置转码任务将转码结果自动投递到谷歌云Pub/Sub消息队列，来触发后续工作流。首先创建一个Pub/Sub的Topic来接收消息。



然后在刚创建的Topic的详情页点击创建Subscription，并选择Create subscription。



给Subscription命名，其它配置保留缺省值即可。这个Subscription可以用来消费消息。



←

Add subscription to topic

A subscription directs messages on a topic to subscribers. Messages can be pushed to subscribers immediately, or subscribers can pull messages as needed.

Subscription ID \*

subscription-transcode-status

?

Subscription name: projects/youzhi-lab/subscriptions/subscription-transcode-status

Topic name

projects/youzhi-lab/topics/transcode-job-status

Delivery type

If Pull, subscribers must request delivery. If Push, Pub/Sub delivers messages as soon as they are published.

☒ Pull

☐ Push

在Subscription详情页，可以点击VIEW MESSAGES按钮，在控制台查看队列中的消息。在转码任务提交并成功完成后，队列中会添加一条新消息，通知任务完成。

←

subscription-transcode-status

EDIT

VIEW MESSAGES

CREATE SNAPSHOT

REPLAY MESSAGES

Subscription details	
Subscription name	projects/youzhi-lab/subscriptions/subscription-transcode-status
Topic name	projects/youzhi-lab/topics/transcode-job-status

消息查看界面如下图。

Messages

Click Pull to view messages and temporarily delay message delivery to other subscribers. Select **Enable ACK messages** and then click **ACK** next to the message to permanently prevent message delivery to other subscribers. Only a few messages will be pulled at a time. Click Pull again to retrieve more messages from the backlog. Use this option cautiously in production environments. If you miss the acknowledgement deadline (10 seconds), the message will be sent again if no other subscribers of this subscription acknowledged the message. [Learn more](#)

PULL

☐ Enable ack messages

Filter table

?

⋮

Publish time	Attribute keys	Message body	Ack ↑
Jan 7, 2021, 6:39:00 PM	—	{\"job\":{\"name\":\"projects/247839977271/locations/asia-east1/jobs/6fbf8d754e327f845ce7f2af00530914\",\"state\":\"SUCCEEDED\",\"failureReason\":null}}	Deadline exceeded ^

# 使用Cloud Functions自动触发转码任务

如果想自动对新上传的视频文件进行转码，使用文件上传触发的Cloud Functions是一个比较合适的选择。下面步骤介绍如何配置一个自动转码的Cloud Functions云函数。

首先创建一个Cloud Functions实例。在配置中选择触发项为对象存储的文件创建事件。

Cloud Functions

Create function

1 Configuration

2 Code

### Basics

Function name \*  
gcs\_start\_transcode

Region  
asia-east2

### Trigger

Cloud Storage

Trigger type  
Cloud Storage

Event type \*  
Finalize/Create

Bucket \*  
hk-publish

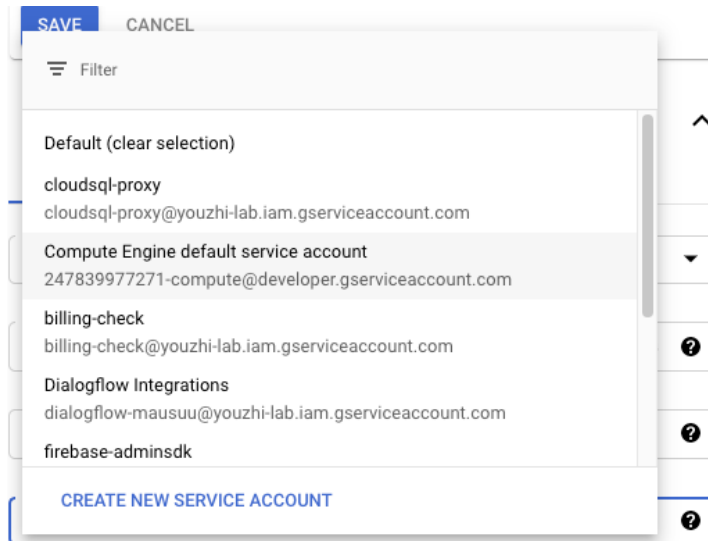
☐ Retry on failure

SAVE

CANCEL

VARIABLES, NETWORKING AND ADVANCED SETTINGS

在Service Account配置中选择可以读写对应对象存储，并执行Transcoder API的服务账号，比如GCE缺省服务账号。



在其它配置中的环境变量配置界面，配置如下几个环境变量。这几个变量会被下面的云函数代码读取，来进行一些自定义的输入输出配置。

VARIABLES, NETWORKING AND ADVANCED SETTINGS

ADVANCED ENVIRONMENT VARIABLES CONNECTIONS

Build environment variables ?

+ ADD VARIABLE

Runtime environment variables ?

Name	Value
PROJECT_ID	youzhi-lab
PROJECT_LOCATION	asia-east1
DEST_LOCATION	gs://video_origin/transcode_output
TEMPLATE_ID	multi-bitrate-20200107

+ ADD VARIABLE

注意函数的触发桶（Trigger Bucket）和转码文件的输出目的桶（DEST\_LOCATION）不要配置成同一个，否则会导致循环转码。

在下面的界面填写Cloud Functions的Python代码和依赖包。

Cloud Functions

Edit function

Configuration

2 Code

Runtime

Python 3.7

▼ ?

Entry point

start\_transcode

Source code

Inline Editor

+

main.py

requirements.txt

...

```
1 from google.auth import compute_engine
2 import google.auth.transport.requests
3 import os
4 import requests
5 import logging
6 from flask import abort
7
8 PROJECT_ID = os.environ['PROJECT_ID']
9 PROJECT_LOCATION = os.environ['PROJECT_LOCATION']
10 DEST_LOCATION = os.environ['DEST_LOCATION']
11 TEMPLATE_ID = os.environ['TEMPLATE_ID']
12
13
14 def start_transcode(event, context):
15
16     bucket_name_input = event['bucket']
17     object_name_input = event['name']
18     file_name = os.path.split(object_name_input)[1]
19     file_name_wo_extension = os.path.splitext(file_name)[0]
20
```

Cloud Functions

Edit function

Configuration

2 Code

Runtime

Python 3.7

▼ ?

Source code

Inline Editor

+

main.py

requirements.txt

...

```
1 # Function dependencies, for example:
2 # package>=version
3 google-auth
4
```

其中，依赖包为

```
google-auth
```

Python代码如下。也可以从

[https://github.com/eugeneyu/cloud-demos/blob/master/cloud-functions/gcs\\_start\\_transcode.py](https://github.com/eugeneyu/cloud-demos/blob/master/cloud-functions/gcs_start_transcode.py) 下载。

```
from google.auth import compute_engine
import google.auth.transport.requests
```

```

import os
import requests
import logging
from flask import abort

PROJECT_ID = os.environ['PROJECT_ID']
PROJECT_LOCATION = os.environ['PROJECT_LOCATION']
DEST_LOCATION = os.environ['DEST_LOCATION']
TEMPLATE_ID = os.environ['TEMPLATE_ID']

def exit_abort():
    return abort(500)

def start_transcode(event, context):

    bucket_name_input = event['bucket']
    object_name_input = event['name']
    file_name = os.path.split(object_name_input)[1]
    file_name_wo_extension = os.path.splitext(file_name)[0]

    cred = compute_engine.credentials.Credentials()

    auth_req = google.auth.transport.requests.Request()
    cred.refresh(auth_req)
    #print(cred.token)

    api_url =
"https://transcoder.googleapis.com/v1beta1/projects/{}/locations/{}/jobs".format(PROJECT_ID,
PROJECT_LOCATION)

    headers = {
        "Authorization": "Bearer {}".format(cred.token),
        "Content-Type": "application/json"
    }

    test = "gs://{}/{}/{}".format(bucket_name_input, object_name_input)
    print("Bucket: {}, inputUri: {}\n".format(bucket_name_input, test))

    data = {
        "inputUri": "gs://{}/{}/{}".format(bucket_name_input, object_name_input),
        "outputUri": "{}/{}/{}".format(DEST_LOCATION, file_name_wo_extension),
        "templateId": "{}".format(TEMPLATE_ID)
    }

    response = requests.post(url = api_url, headers=headers, json = data)

    if not response.ok or "error" in response.text:
        logging.error(RuntimeError(response.text))
        exit_abort()

    response_json = response.json()

```

```
print("New job started - Job Name: {}".format(response_json['name']))
```

部署完云函数后，可以往输入桶上传一个视频文件，然后在云函数详情页查看执行日志。转码任务提交完成后，可以到Pub/Sub界面查看转码完成通知。也可以到对象存储响应输出目录查看结果文件。也可以用浏览器或[5] [Shaka播放器](#)测试播放。

## 使用Cloud Functions自动发布转码后视频

与上一节步骤类似，可以新建一个Cloud Functions云函数，设置其触发源为Pub/Sub消息队列。根据转码任务的完成消息，可以进行将输出文件移动到发布路径，并更新媒资系统或内容数据库。如果消息队列中收到的是转码失败消息，则做相应的错误处理和通知，比如将源视频文件移动到单独的目录。对于这些处理本文不提供示例代码，但是可以参考以下移动对象存储中文件的云函数。

[https://github.com/eugeneyu/cloud-demos/blob/master/cloud-functions/bucket\\_to\\_bucket.py](https://github.com/eugeneyu/cloud-demos/blob/master/cloud-functions/bucket_to_bucket.py)

# 附录

## A. 参考文档和工具

- [1] [GCP Transcoder API概览](#)
- [2] [GCP Transcoder API Quickstart](#)
- [3] [JobConfig reference](#)
- [4] [Installing Google Cloud SDK](#)
- [5] [Shaka播放器](#)

## B. Job Template示例

```
{
  "config": {
    "inputs": [
      {
        "key": "input0",
      }
    ],
    "editList": [
      {
        "key": "atom0",
        "inputs": [
          "input0"
        ]
      }
    ],
    "elementaryStreams": [
      {
        "key": "video-stream0",
        "videoStream": {
          "codec": "h264",
          "heightPixels": 360,
          "widthPixels": 640,
          "bitrateBps": 400000,
          "frameRate": 50,
          "enableTwoPass": true,
          "gopDuration": "5s",
          "vbvSizeBits": 18000000,
        }
      },
      {
        "key": "video-stream1",
        "videoStream": {
          "codec": "h264",
          "heightPixels": 480,
```

```
        "widthPixels": 854,
        "bitrateBps": 700000,
        "frameRate": 50,
        "enableTwoPass": true,
        "gopDuration": "5s",
        "vbvSizeBits": 18000000,
    }
},
{
    "key": "video-stream2",
    "videoStream": {
        "codec": "h264",
        "heightPixels": 720,
        "widthPixels": 1280,
        "bitrateBps": 1400000,
        "frameRate": 50,
        "enableTwoPass": true,
        "gopDuration": "5s",
        "vbvSizeBits": 18000000,
    }
},
{
    "key": "video-stream3",
    "videoStream": {
        "codec": "h264",
        "heightPixels": 1080,
        "widthPixels": 1920,
        "bitrateBps": 2100000,
        "frameRate": 50,
        "enableTwoPass": true,
        "gopDuration": "5s",
        "vbvSizeBits": 18000000,
    }
},
{
    "key": "video-stream4",
    "videoStream": {
        "codec": "h264",
        "heightPixels": 1080,
        "widthPixels": 1920,
        "bitrateBps": 4200000,
        "frameRate": 50,
        "enableTwoPass": true,
        "gopDuration": "5s",
        "vbvSizeBits": 18000000,
    }
},
{
    "key": "audio-stream0",
    "audioStream": {
        "codec": "aac",
        "bitrateBps": 64000
    }
}
```



```
    }
  }
],
"muxStreams": [
  {
    "key": "400-kbps-ts",
    "container": "ts",
    "elementaryStreams": [
      "video-stream0",
      "audio-stream0"
    ],
    "segmentSettings": {
      "segmentDuration": "5s",
      "individualSegments": true
    }
  },
  {
    "key": "700-kbps-ts",
    "container": "ts",
    "elementaryStreams": [
      "video-stream1",
      "audio-stream0"
    ],
    "segmentSettings": {
      "segmentDuration": "5s",
      "individualSegments": true
    }
  },
  {
    "key": "1400-kbps-ts",
    "container": "ts",
    "elementaryStreams": [
      "video-stream2",
      "audio-stream0"
    ],
    "segmentSettings": {
      "segmentDuration": "5s",
      "individualSegments": true
    }
  },
  {
    "key": "2100-kbps-ts",
    "container": "ts",
    "elementaryStreams": [
      "video-stream3",
      "audio-stream0"
    ],
    "segmentSettings": {
      "segmentDuration": "5s",
      "individualSegments": true
    }
  }
],
```

```
{
  "key": "4200-kbps-ts",
  "container": "ts",
  "elementaryStreams": [
    "video-stream4",
    "audio-stream0"
  ],
  "segmentSettings": {
    "segmentDuration": "5s",
    "individualSegments": true
  }
},
],
"manifests": [
  {
    "fileName": "master.m3u8",
    "type": "HLS",
    "muxStreams": [
      "400-kbps-ts",
      "700-kbps-ts",
      "1400-kbps-ts",
      "2100-kbps-ts",
      "4200-kbps-ts"
    ]
  }
],
"output": {
}
}
```