Samsung Electronics

Intelligent Video Processing at Scale

With Disco Parallelization

Table of Contents

Copyright

For more information, please visit http://developer.samsung.com

Trademarks and Service Marks

Document History

Version Number	Date	Editor	Scope of Update
0.0	2019/09/08	Zohar Sacks	Created document
0.1	2019/09/15	Jon Barker & Raymond Lo	Added Tutorial & Solution Section
2	2019/09/15	Raymond Lo	Draft with full lesson documentations
2.1	2019/09/23	Lanz Laqui	Document review
3.0	2019/9/24	Raymond Lo	Final Draft
3.1	2019/10/1	Raymond Lo	Updated content bugs

Overview

We knew there was something more to the cloud, so we went and built it.
Disco is the next generation of serverless computing. Whether your jobs are periodic or ongoing, you need responsive compute capable of handling your most demanding workloads. Disco scales on-demand to expedite results and reduce the costs related to compute-intensive tasks such as Natural Language Processing (NLP), ETL, Image and Video Processing, Gene Sequencing, and more.

Our mission is to create a simple, secure, scalable computing service based on the following three principles:

- **Optimize Use of Existing Resources**
  Why build more data centers and use more energy when we can save Earth's resources by tapping into the surplus of today's existing technology?

- **Remove Complexity**
  Why complicate data processing when we can provide a simple, intuitive service to expedite the processes you want to execute?

- **Grow Serverless Computing**
  Why worry about managing servers and capacity when we can manage everything for you and provide faster results?

Objectives

We will write and run a full-blown Python image processing application, scaling it using Disco. Each participant will have time to experiment with the Disco platform, learning how to run compute-intensive jobs and parallelize them with ease.

The participant will also experiment with disco parallelization platform in a fun yet educating way.

Setup

To run the code, the following software is required to be installed on your local machine.

1. **Docker Desktop**
   http://www.docker.com

   Please note that a user account should be created from:
   a. To install on Mac, please refer to this link.
      https://docs.docker.com/docker-for-mac/install/
   b. To install on Windows, please refer to this link.
      https://docs.docker.com/docker-for-windows/install/
   c. To install on Linux (Ubuntu, CentOS, or others), please refer to this link.
      https://docs.docker.com/install/linux/docker-ce/ubuntu/

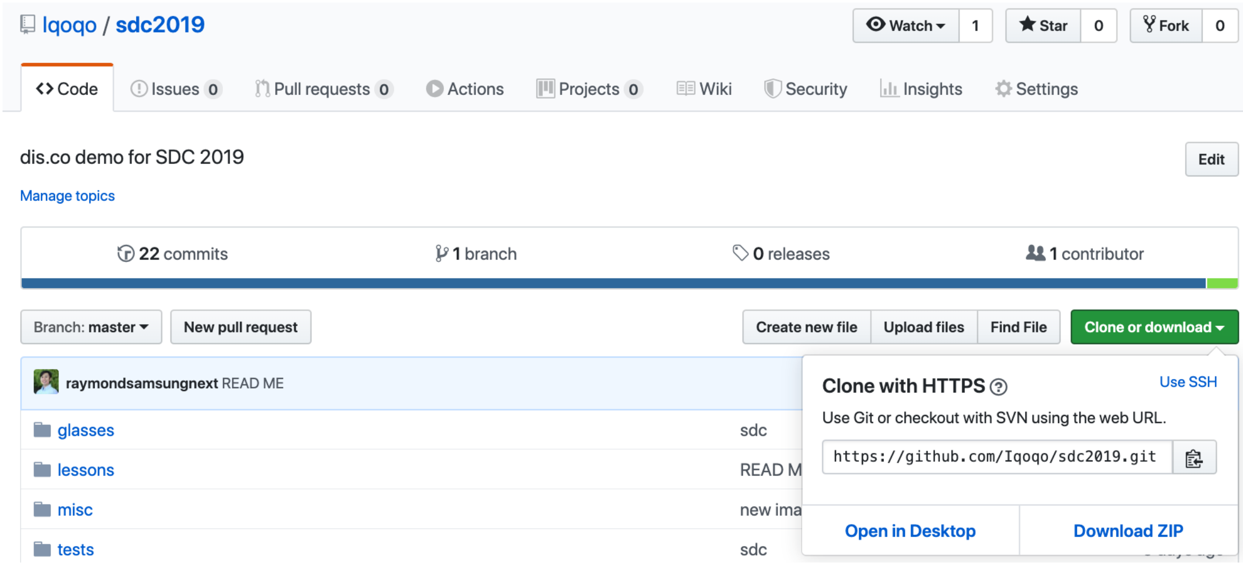
2. **Git** (optional but recommended)
   a. Windows
      https://git-scm.com/download/win
   b. Mac or Linux
      https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Then, the source code of the lessons can be downloaded from the GitHub repository.
https://github.com/Iqoqo/sdc2019

To download the code, you can either use the following command with a command-line interface.

```
git clone https://github.com/Iqoqo/sdc2019
```

Or, we can download the zip file directly from the GitHub repository. Once you have downloaded the file (as a zip file), please extract the file to your own favourite location. We will refer the path to the directory as the ***project root*** in the future.



Lesson 1 - Setup the Docker Environment

Docker is a convenient and easy way to standardize an environment in a very lightweight Linux environment (usually Alpine Linux running in the Docker Engine) that contains all dependencies required to run a program. In this case, the docker image provided is preconfigured to include disco, discomp (disco's parallelization multi-processes API), and other image processing and quantitative libraries such as the cv2 (computer vision library), FFmpeg (video processing library), and numpy (scientific computing library).

The purpose of this lesson is to verify that the Docker Environment is setup correctly on the machine.

Step 1: Download the disco docker image with the following command line.

```
docker pull iqoqo/discofy:sdc.local
```

```
(env) Raymonds-MacBook-Pro:Documents raymondlo84$ docker pull iqoqo/discofy:sdc.local
sdc.local: Pulling from iqoqo/discofy
1ab2bdfe9778: Pull complete
b5d689d9c40c: Pull complete
5b13ee99f0ea: Pull complete
d617973d7fa5: Pull complete
abfef9fe6f0b: Pull complete
d6650b4cf828: Pull complete
51d0112d14a3: Pull complete
aecbac94b60d: Pull complete
28a0391d57fb: Pull complete
f59ecbd44ad3: Pull complete
7be92546b80f: Pull complete
a63cbb7b520a: Pull complete
9d03398a34f6: Pull complete
86001cc80bc9: Pull complete
e3aa8d2963b7: Pull complete
09627cd030eb: Pull complete
c7426c2121fe: Pull complete
62590d74f0d2: Pull complete
1f391a55b64a: Pull complete
be660cdf6402: Pull complete
1579a7abf115: Pull complete
Digest: sha256:cba9b475aa8c89c31f3c89c43784920ee7ee0d9413896cc5ad11a68435c869b6
Status: Downloaded newer image for iqoqo/discofy:sdc.local
docker.io/iqoqo/discofy:sdc.local
```

Step 2: Tag the docker image with the following command line.

```
docker tag iqoqo/discofy:sdc.local discofy:sdc.local
```

Step 3: Check the image is installed in docker.

```
docker images
```

```
(env) Raymonds-MacBook-Pro:Documents raymondlo84$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
iqoqo/discofy       sdc.local           9d0e4e1c1233        26 hours ago        1.6GB
discofy             sdc.local           9d0e4e1c1233        26 hours ago        1.6GB
```

Step 4: In the project root directory (e.g., `/Users/raymondlo84/Documents/sdc2019`), execute the following command line to run the script in the docker environment.

```
docker run -it -v `pwd`:/home/codelab/ -w /home/codelab/lessons/lesson_1/ discofy:sdc.local pytest
```

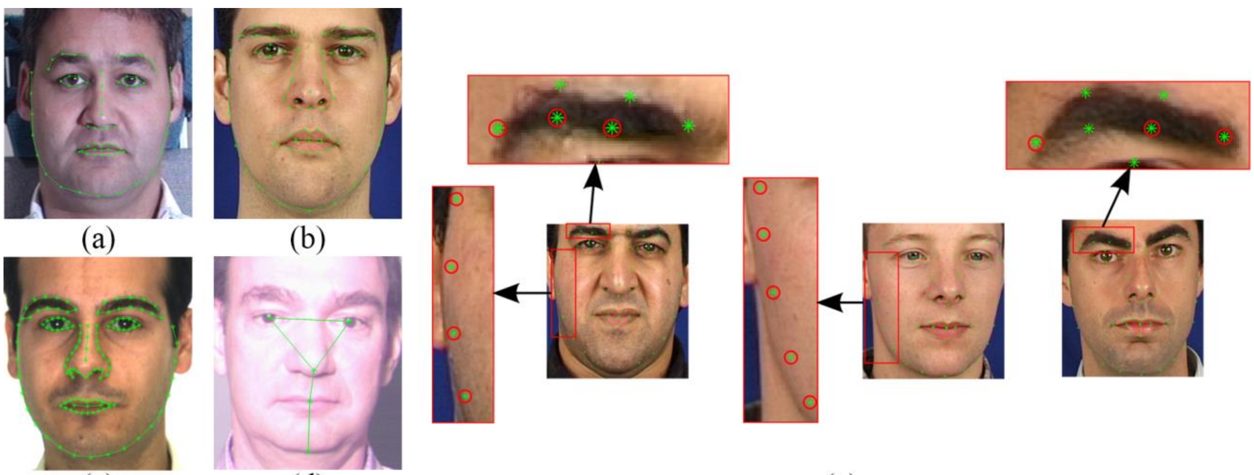
```
(env) Raymonds-MacBook-Pro:sdc2019 raymondlo84$ docker run -it -v `pwd`:/home/codelab/ -w /home/codelab/lesso
ns/lesson_1/ discofy:sdc.local pytest
============================================= test session starts =============================================
platform linux -- Python 3.7.4, pytest-5.0.0, py-1.8.0, pluggy-0.13.0
rootdir: /home/codelab/lessons/lesson_1
collected 2 items

tests/test_1.py ..                                                                                      [100%]

============================================== warnings summary ===============================================
tests/test_1.py::test_all_imports
  /usr/local/lib/python3.7/site-packages/past/translation/__init__.py:35: DeprecationWarning: the imp module
is deprecated in favour of importlib; see the module's documentation for alternative uses
    import imp

tests/test_1.py::test_all_imports
  /usr/local/lib/python3.7/site-packages/past/types/oldstr.py:5: DeprecationWarning: Using or importing the A
BCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
    from collections import Iterable

tests/test_1.py::test_all_imports
  /usr/local/lib/python3.7/site-packages/past/builtins/misc.py:4: DeprecationWarning: Using or importing the
ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
    from collections import Mapping

-- Docs: https://docs.pytest.org/en/latest/warnings.html
=================================== 2 passed, 3 warnings in 0.23 seconds ====================================
```

The result above show that the docker had successfully executed the `pytest` script.

Lesson 2 - Discofy an image

In lesson 1, we provided a Docker environment that runs Python code with OpenCV, FFmpeg, and other scientific libraries such as numpy. In this lesson, we will provide an interesting example of how we can overlay eyeglasses onto a face detected in an image using these libraries.

Additionally, we have also added the Dlib library for supporting the face detection and face feature extraction (facial landmarks) based on machine learning. In our code, a pre-trained facial landmark detector and model (i.e., `shape_predictor_68.dat`) are added to estimate the location of 68 coordinates that map to facial structures on the detected face.



(a) (b) (c) (d) (e)

For further details on how the facial landmark works, please refer to https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

Step 1: Open `glasses_2.py` in the `lessons/lesson_2` directory.

```python
def discofy_image(in_image_path, out_image_path):

    # use cv2 to read the image and glasses to memory
    # HINT: glasses image is a png with alpha channel.
    #       You'll need to use cv2.IMREAD_UNCHANGED flag

    # use getters to get detector and predictor

    # use mesh_overlays to put shades on the image

    # delete the output file if it exists
    try:
        os.remove(out_image_path)
    except FileNotFoundError:
        pass

    # use cv2 to write results
```

Hint: Look at `mesh_overlays` and `overlay_transparent`, these are the main functions that we use for the face detection and meshing of the images.

Step 2: Add in the solution code to `discofy_image.`

```python
def discofy_image(in_image_path, out_image_path):
    img = cv2.imread(in_image_path)
    shades_overlay = cv2.imread("disco_glasses.png", cv2.IMREAD_UNCHANGED)

    detector = get_detector()
    predictor = get_predictor()

    out_frame = mesh_overlays(img, shades_overlay, detector, predictor)
    try:
        os.remove(out_image_path)
    except FileNotFoundError:
        pass

    cv2.imwrite(out_image_path, out_frame)
```

Step 3: To run the solution, execute the following command in the ***project root*** directory.

```
docker run -it -v `pwd`:/home/codelab/ -w /home/codelab/lessons/lesson_2/
discofy:sdc.local python glasses_2.py <input image path> <output image path>
```

Replace `<input image path>` and `<output image path>` with your own image path. In this example, we have provided an image of dis.co team in the `lessons/lesson_2/team.png` directory.

Step 4. Run the tester to generate and validate the results.

```
docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/lessons/lesson_2
discofy:sdc.local pytest
```

Step 5. Open the result images located in `lessons/lesson_2/disco-team1.jpg`

Lesson 3 - Discofy videos

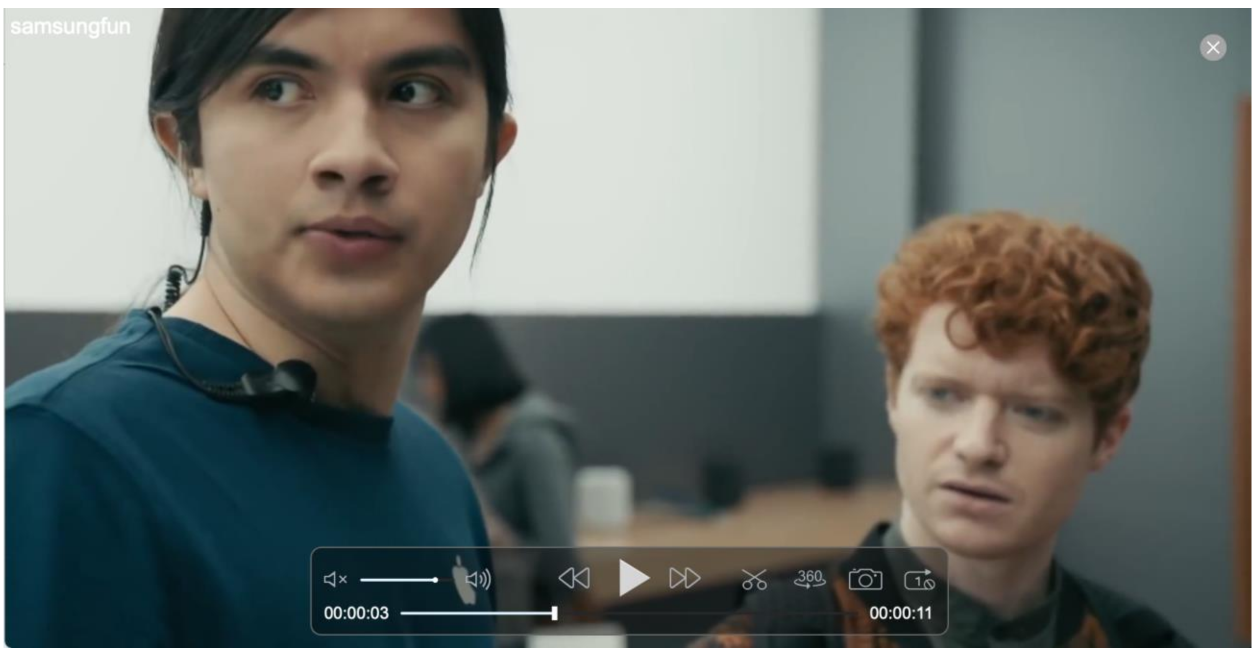
Now, we have acquired a powerful tool to intelligently process and overlay our 'disco shade' onto a face detected in an image based on machine learning. The next natural step is how we can scale this to a video (i.e., a sequence of images) or even on hundreds of different streams of videos.

To handle video files, the FFmpeg library provided a very simple and easy-to-use interface to extract image frames from a video. Similarly, we can also create a video with image sequences with the same library.

To get started, we will first look into the `lessons/lesson_3` folder.

```
Raymonds-MacBook-Pro:lesson_3 raymondlo84$ ls
README.md            __pycache__          glasses_3.py            lessons              team.png
__init__.py          disco_glasses.png    glasses_3_solution.py   samsungfun.mp4       tests
```

As you can see, we have now provided the `glasses_3.py` and a short video clip called `samsungfun.mp4` for you to test the work on.



Now, let's start by looking into the source code itself.

Step 1: Open `lessons/lesson_3/glasses_3.py`. This time the implementation of the `discofy_video` function is left to be completed. To simplify the work, we have already provided the methods supported by OpenCV for extracting frames from a video.

```python
def discofy_video(in_video_path, out_video_path):
    """
    Discofy a video.
    :param in_video_path: Path to input video
    :param out_video_path: Path to target output video. It is user's
            responsibility to make sure the target directory exists.
            If a file with the same name exists it will be overwritten.
    :return: None
    """
    detector = get_detector()
    predictor = get_predictor()

    # source video
    cap = cv2.VideoCapture(in_video_path)
    source_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    source_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    source_fps = int(cap.get(cv2.CAP_PROP_FPS))

    # video writer
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(INTERIM_VIDEO,
                          fourcc,
                          source_fps,
                          (source_width, source_height))

    # We want the shades to be animated. So we prepared 12 images of the shades
    # with different glare. Following are the paths to all images.
    # In the video in frame i we want to take shades image i%12.
    # Don't forget to read images to memory with cv2.imread
    shades_paths = [abs_path(f"glasses/disco_glasses_{i:02d}.png")
                    for i in range(1, 13)]

    # loop over all images of the capture video.
    # Hint: ret, frame = cap.read().
    #        ret is true if the next frame is a valid image and false otherwise
    # Notice: In the last cap.read() ret is False. Ypu still have to write
    #          that frame to out to indicate the end of the video but you
    #          shouldn't do any image processing on it

    # while ???:
    #     read the next frame

    #     if not ret:
    #         out.write(frame)
    #         break

    #     find the right shades image
    #     mesh the shades to the image (use mesh_overlays)
    #     write the result image to the out video (Hint: out.write(frame))

```

Step 2: Replace the code from line 247 onwards with the following code.

```python
    frame_count = 0
    overlays = overlay_iterator()

    while cap.isOpened():
        ret, frame = cap.read()

        if not ret:
            out.write(frame)
            break

        shades_overlay = next(overlays)
        frame_count += 1
        out_frame = mesh_overlays(frame, shades_overlay, detector, predictor)
        out.write(out_frame)

    # Release capture devices
    cap.release()
    out.release()

    in1 = ffmpeg.input(INTERIM_VIDEO)
    in2 = ffmpeg.input(in_video_path)
    v1 = in1.video
    a2 = in2.audio
    out = ffmpeg.output(v1, a2, out_video_path)
    out.run(overwrite_output=True)
```

The code above basically read each frame from the input video and perform the `mesh_overlays` function discussed in Lesson 2. Then, the result is written out with the FFmpeg function. The reason that we have to perform two streams `in1` and `in2` is because we would like to preserve the audio stream from the original video.

Step 3: To test the code, run the code on Docker with the following command with the desired video.

```
docker run -it -v `pwd`:/home/codelab/ -w /home/codelab/lessons/lesson_3/
discofy:sdc.local python glasses_3.py <input video path> <output video path>
```

The <input video path> and <output video path> should be replaced by the appropriate file path such as `lessons/lesson_3/samsungfun.mp4`.

Step 4: Run the test code and generate our test results.

```
docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/lessons/lesson_3
discofy:sdc.local pytest
```

Step 5. Review the result from `lessons/lesson_3/disco-samsungfun1.mp4`

Lesson 4 - Disco CLI

In the last two lessons, we have created pipelines to process images or videos with some of the state-of-the-art libraries and machine learning algorithms. However, as we may see quickly that these processes take a long while to analyze video footage, especially with high definition images and videos.

Here we introduce the disco command-line interface (CLI). This is one of the easiest way to get familiar with the disco interfaces that allow us to scale and offload the work to a dedicated cloud resource of your choice.

Step 1: Run docker interactively. The terminal now is running under the Docker machine.

```
docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/lessons/lesson_4
discofy:sdc.local
```

```
(env) Raymonds-MacBook-Pro:sdc2019 raymondlo84$ docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/less
ons/lesson_4 discofy:sdc.local
root@f6f46e57d5ea:/home/codelab/lessons/lesson_4#
```

Step 2: In the same terminal from the last step, we run the following command. This will return the help menu from the disco CLI with a bit of ASCII art from disco.

```
disco -h
```

```
dis.co command line tool
-----------------------------
This tool can be used to create, view, start and cancel jobs on the disco platform.

Options:

  --debug              Debug mode
  -h, --help           output usage information

Commands:

  add [options]        Submit a new job
  listc                List available clusters
  login [options]      Login to Dis.co with your credentials
  logout               Logout of Dis.co
  start [options]      Start a specific job
  list [options]       List all jobs
  jobSummary           Summary all jobs
  view [options]       View specific job details (-d to download)
  cancel [options]     Cancel a specific job
  archive [options]    Archive a specific job
  version              Show installed version info
```

Now, let’s explore some of the features in the CLI. In the `hello_disco.py`, there is a simple print statement, and in the next step we will run the code on the disco cloud servers.

Step 3. Run the `hello_disco.py` with disco CLI.

```
disco add --name hello_job --script hello_disco.py --wait --run
```

```
root@70fe4ea71917:/home/codelab/lessons/lesson_4# disco add --name hello_job --script hello_disco.py --wait --run
  Uploading script file(s)...
Warning:  Python virtual environment was not detected. Dis.co will not ba able to automatically update your cloud
Uploaded all assets.
  Submitting job hello_job...Job created successfully!
jobId: [5d7f6f898d9278000a5064e2]
Job started successfully!

hello_job (Job ID: 5d7f6f898d9278000a5064e2)

Status: Done

   # # # # #
 # # # # # # #
 # #       # #
 # #       # #
 # #       # #
 # # # # # # #
   # # # # #

#  Success: 1 (100%)

Tasks:
┌───┬──────────────────────────┬─────────┬─────────────┬──────────┐
│ # │ ID                       │ Status  │ Last Change │ Runtime  │
├───┼──────────────────────────┼─────────┼─────────────┼──────────┤
│ 1 │ 5d7f6f898d9278000a5064e3 │ Success │ just now    │ 00:00:01 │
└───┴──────────────────────────┴─────────┴─────────────┴──────────┘

Activities Log:
 Created a minute ago
 Started a minute ago
```

Notice the job_id `[5d7f6f898d9278000a5064e2]`, this unique ID is going to be needed for checking the status or to download the result from the job.

Step 5: To download results from the execution, run the following command

```
disco view --job <your_job_id> --download
```

Again, replace `<your_job_id>` with the appropriate job id returned by the process. Lastly, we can examine the results by extracting the downloaded results.

```
root@70fe4ea71917:/home/codelab/lessons/lesson_4# disco view --job 5d7f6f898d9278000a5064e2 --download

hello_job (Job ID: 5d7f6f898d9278000a5064e2)

Status: Done


    # # # # #
  # # # # # # #
  # #       # #
  # #       # #
  # #       # #
  # # # # # # #
    # # # # #

#  Success: 1 (100%)


Tasks:
+---+--------------------------+---------+-------------+----------+
| # | ID                       | Status  | Last Change | Runtime  |
+---+--------------------------+---------+-------------+----------+
| 1 | 5d7f6f898d9278000a5064e3 | Success | just now    | 00:00:01 |
+---+--------------------------+---------+-------------+----------+

Activities Log:
 Created 10 minutes ago
 Started 10 minutes ago

Successfully downloaded task and its results into '5d7f6f898d9278000a5064e2-hello_job'!
```

Step 6: Extract the zip file with this command

```
unzip <path to downloaded>
```

In our case, the result can be found in the stdout file.

```
root@70fe4ea71917:/home/codelab/lessons/lesson_4# unzip 5d7f6f898d9278000a5064e2-hello_job/results/5d7f6f898d9278000a5064e3.zip
Archive:  5d7f6f898d9278000a5064e2-hello_job/results/5d7f6f898d9278000a5064e3.zip
  inflating: IqoqoTask.stdout.0.txt
  inflating: IqoqoTask.stderr.0.txt
```

Step 7: Display the output result.

```
root@70fe4ea71917:/home/codelab/lessons/lesson_4# cat IqoqoTask.stdout.0.txt
Hello Disco
```

That's it. We have just completed our first cloud-based execution without changing a single line of code.

Lesson 4.1 - Using Disco to Perform Image Processing on the Cloud

Lesson 2 demonstrates a computer vision example of detecting faces, facial landmarks and overlaying virtual eyeglasses onto a person's face. In this lesson, we will offload the exact work to the Disco Cloud and perform such remotely with only a few lines of code changes. More importantly, with the Disco's serverless architecture, you can scale this to 100 or 1000 times without managing the resources or making any code changes on developer's end.

In this demo, we have created a customized docker image (with OpenCV, FFmpeg, and Dlib libraries pre-installed) that deploys by each agent running on the server. If desired we can also setup GPU resources per server and scale even further for your jobs.

The Python script `glasses_4.py` provides a full-blown example of how Disco handles input and output on the server side.

Particularly, the Disco command takes the script `glasses_4.py` and the input image team.jpg, upload to the cloud server, and perform the processing remotely. During the executions, any data output to the '/local/run-result' folder on the server side will be saved in our final result. Upon the success of the execution, Disco packages the results in `<job id>-<job name>` folder, and we download the package back on the client side.

Step 1. Run Docker interactively on the terminal. At the project root directory, we run the following command.

```
docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/lessons/lesson_4.1
discofy:sdc.local
```

Step 2. Run the disco command to upload and run the job on the Disco cloud.

```
disco add --name disco_image --script glasses_4.py --input team.jpg --wait --
run --download
```

```
root@5799168df5d9:/home/codelab/lessons/lesson_4.1# disco add --name disco_image --script glasses_4.py --input team.jpg --wai
t --run --download
  ⣿ Uploading input file(s)....
Warning:  Python virtual environment was not detected. Dis.co will not ba able to automatically update your cloud environment
 when not using python's virtual environment.
Uploaded all assets.
  ⣿ Submitting job disco_image...Job created successfully!
jobId: [5d939869614056000c662f74]
Job started successfully!

disco_image (Job ID: 5d939869614056000c662f74)

Status: Done

    # # # # #
  # # # # # # #
  # #       # #
  # #       # #
  # #       # #
  # # # # # # #
    # # # # #

#  Success: 1 (100%)

Tasks:
┌───┬──────────────────────────┬─────────┬─────────────┬──────────┐
│ # │ ID                       │ Status  │ Last Change │ Runtime  │
├───┼──────────────────────────┼─────────┼─────────────┼──────────┤
│ 1 │ 5d939869614056000c662f75 │ Success │ just now    │ 00:00:03 │
└───┴──────────────────────────┴─────────┴─────────────┴──────────┘

Activities Log:
 Created a minute ago
 Started a minute ago

Successfully downloaded task and its results into '5d939869614056000c662f74-disco_image'!
```

Step 3. Extract the results (with either GUI interface or unzip command in terminal)

```
unzip <job id>-<job name>/results/<job id>.zip
```

As usual, we will replace <job id> and <job name> accordingly.

```
root@5799168df5d9:/home/codelab/lessons/lesson_4.1/5d939869614056000c662f74-disco_image/results# unzip 5d939869614056000c662f
75.zip
Archive:  5d939869614056000c662f75.zip
 extracting: disco_team.jpg
  inflating: IqoqoTask.stdout.0.txt
  inflating: IqoqoTask.stderr.0.txt
```

The final result should match exactly what we see in Lesson 2, and that's it.

Lesson 5 - Using Disco to perform Video Processing on the Cloud

With the disco platform, we can now send jobs across an unlimited number of resources (devices) without the pain of managing them. Basically, compute on-demand and scale as needed. In this part of the lesson, we will demonstrate how to parallelize the video processing pipeline by splitting the work into smaller clips that can be offloaded directly to disco. For example, we can divide up a ~4 minute video into 10-second clips, and process each clip in parallel and merge the results in a later step. Again, there is a trade-off on the granularity of the parallelization and it really depends on the use cases. Developers can examine this further by test running the solution with a small scale test.

Now, let's explore how we can split videos with FFmpeg, and then how we can offload tasks to disco and merge the results back.

Step 1. Run the Docker interactively.

```
docker run -it -v `pwd`:/home/codelab/  -w /home/codelab/lessons/lesson_5
discofy:sdc.local
```

Step 2. In the terminal, run the FFmpeg command to split the video into 10 second chucks.

```
ffmpeg -i samsung10.mp4 -c copy -map 0 -segment_time 00:00:10 -f segment
split%03d.mp4
```

This command generates a set of 10 second clips as `split001.mp4`, `split002.mp4`, etc…

Step 3. Create disco jobs on each video clip and process them remotely.

```
disco add --name parallel_video --script glasses_5.py --wait --run --download
--input "split*mp4"
```

```
root@1c0f82076ee1:/home/codelab/lessons/lesson_5# disco add --name parallel_video --script glasses_5.

  ⣾ Uploading input file(s)....
Warning:  Python virtual environment was not detected. Dis.co will not ba able to automatically updat
Uploaded all assets.
  ⣾ Submitting job parallel_video...Job created successfully!
jobId: [5d7f7f17fc4015000dd1cdf8]
Job started successfully!

parallel_video (Job ID: 5d7f7f17fc4015000dd1cdf8)

Status: Done


    # # # # #
  # # # # # # #
  # #       # #
  # #       # #
  # #       # #
  # # # # # # #
    # # # # #

#  Success: 7 (100%)


Tasks:
```

#	ID	Status	Last Change	Runtime
1	5d7f7f17fc4015000dd1cdfa	Success	just now	00:01:46
2	5d7f7f17fc4015000dd1cdf9	Success	just now	00:01:52
3	5d7f7f17fc4015000dd1cdfd	Success	just now	00:01:58
4	5d7f7f17fc4015000dd1cdfb	Success	just now	00:01:54
5	5d7f7f17fc4015000dd1cdfc	Success	just now	00:01:46
6	5d7f7f17fc4015000dd1cdfe	Success	just now	00:01:37
7	5d7f7f17fc4015000dd1cdff	Success	just now	00:00:24

Each job ran for approximately 90 seconds on the disco resources. Each input (i.e., the video clips) is automatically distributed across the resources. On the developer side, we have zero lines of code changed to get these batches to run in parallel. Yes, it's amazing.

Upon completion of these tasks, the disco CLI downloads the results back to the local machine because we provided the `--download` flag in the last command.

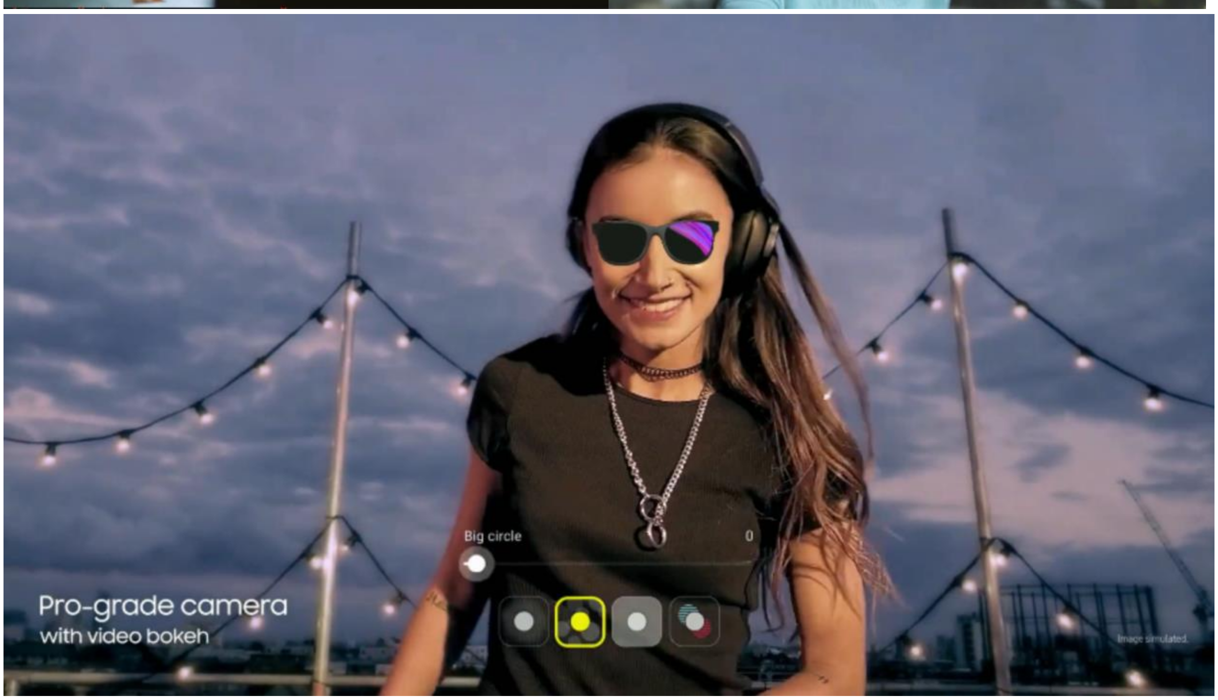
```
Activities Log:
 Created 5 minutes ago
 Started 5 minutes ago

Successfully downloaded task and its results into '5d7f7f17fc4015000dd1cdf8-parallel_video'!
```

Step 4. Run the commands to merge the results back with FFmpeg and some bash scripts.

```
# unzip results
cd *parallel_video/results &&
for f in `ls *.zip`; do unzip -o $f; done
```

```
# write the parts to a manifest
rm -f manifest.txt;
for f in `ls *.mp4`; do echo "file '$f'" >> manifest.txt; done

# stitch back the video
ffmpeg -f concat -safe 0 -i manifest.txt -c copy discofy_toy.mp4

# move back the result to the lesson dir
mv discofy_toy.mp4 ../..

# go back home
cd ../..
```

Now, we have it, the final result can be found in `lessons/lesson_5/disco-samsung.mp4`



Bonus - discomp

Disco also supports multiprocessing API natively with Python. That means if you have prior experiences working with the Python MP, we can easily port the code to support MP with a single line of code changing.

This bonus lesson will demonstrate an analytics example of processing YouTube videos with the discoMP APIs that work natively with your Python code.

Step 1: Learn the code by reading `you_tube_face_detection.py` located in `lessons/lessons_6`.

```python
def handle_url(url):

    try:
        video_file = download_from_you_tube(url)
    except urllib.error.HTTPError as e:
        print(f'{e} {url}')
        return [0], 1
    except KeyError as e:
        print(f'{e} {url}')
        return [0], 1
    except pytube.exceptions.VideoUnavailable as e:
        print(f'{e} {url}')
        return [0], 1
    except:
        print('Unknown error')
        print(f' {url}')
        return [0], 1

    faces_time_series, frame_rate = video_face_detection(video_file)
    return faces_time_series, frame_rate
```

The function `handle_url` is the entry point to analyze a video from a YouTube video URL. In the initial example, we provide a single-thread example that process these videos one at a time.

```python
def main_single_thread():
    urls = parse_args()
    for uid, url in enumerate(urls):
        faces_time_series, frame_rate = handle_url(url)
        plot_faces(faces_time_series, frame_rate, uid)
```

However, there are various ways we can parallelize it in Python. Particularly, we will utilize the Pool object from the multiprocessing Python module.

```
def handle_urls_mp(urls):
    start = time.time()
    print(f"multi process {len(urls)} urls")
    thread_pool = Pool()
    res = thread_pool.map(handle_url, urls)
    end = time.time()
    print(f'multi process {len(urls)} urls took {end - start} sec')
    return res
```

Please refer to https://docs.python.org/3/library/multiprocessing.html for additional documentation.

Step 2: Run each MP version with disco

There are 4 versions of the `main` function:

- `main_single_thread` - a single-threaded main
- `main_mp` - a multi-process version running locally on your computer
- `main_discomp` - a multi-process version where each process handles a single video remotely on disco cloud
- `main_discomp_mp` - a multi-process version where each process handles multiple videos (in parallel) remotely on disco cloud

To run each version, we can simply comment and uncomment the relevant code in the main function.

```
if __name__ == "__main__":
    main_start = time.time()
    main_single_thread()
    #main_mp()
    #main_discomp()
    #main_discomp_mp()
    main_end = time.time()
    print(f'main compute took {main_end - main_start} sec')
```

Show us your results and let us know how much performance gain you have squeezed out from dis.co :). This is it for our secret bonus mission. Thank you for coding.