googlecoursera.qwiklabs.com

Automate updating catalog information | Qwiklabs

Qwiklabs

21-27 minutes

Introduction

You work for an online fruits store, and you need to develop a system that will update the catalog information with data provided by your suppliers. The suppliers send the data as large images with an associated description of the products in two files (.TIF for the image and .txt for the description). The images need to be converted to smaller jpeg images and the text needs to be turned into an HTML file that shows the image and the product description. The contents of the HTML file need to be uploaded to a web service that is already running using Django. You also need to gather the name and weight of all fruits from the .txt files and use a Python request to upload it to your Django server.

You will create a Python script that will process the images and descriptions and then update your company's online website to add the new products.

Once the task is complete, the supplier should be notified with an email that indicates the total weight of fruit (in lbs) that were uploaded. The email should have a PDF attached with the name of the fruit and its total weight (in lbs).

Finally, in parallel to the automation running, we want to check the health of the system and send an email if something goes wrong.

What you'll do

- Write a script that summarizes and processes sales data into different categories
- Generate a PDF using Python
- Automatically send a PDF by email
- Write a script to check the health status of the system

You'll have 120 minutes to complete this lab.

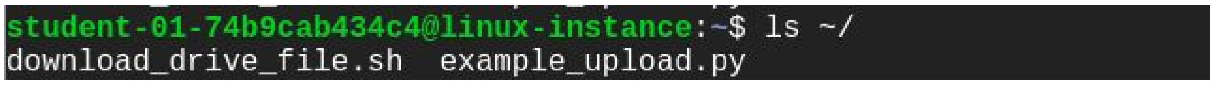
Fetching supplier data

You'll first need to get the information from the supplier that is currently stored in a Google Drive file. The supplier has sent data as large images with an associated description of the products in two files (.TIF for the image and .txt for the description).

Here, you'll find two script files download_drive_file.sh and the example_upload.py files. You can view it by using the following command.

```
ls ~/
```

Output:



```
student-01-74b9cab434c4@linux-instance:~$ ls ~/
download_drive_file.sh  example_upload.py
```

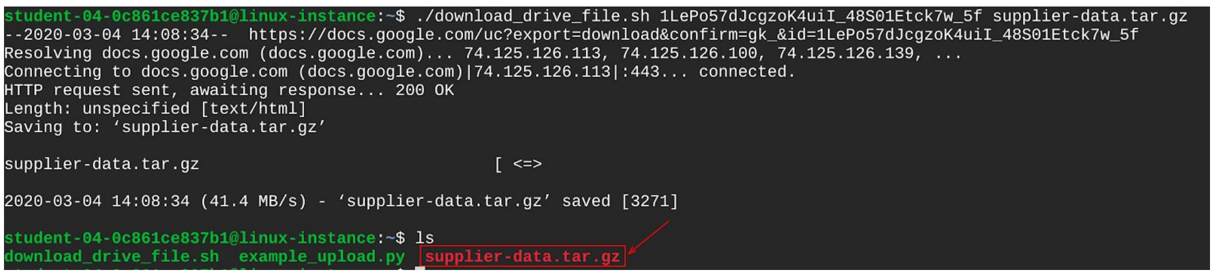
To download the file from the supplier onto our linux-instance virtual machine we will first grant executable permission to the download_drive_file.sh script.

```
sudo chmod +x ~/download_drive_file.sh
```

Run the download_drive_file.sh shell script with the following arguments:

```
./download_drive_file.sh 1LePo57dJcgzoK4uiI_48S01Etck7w_5f
supplier-data.tar.gz
```

Output:

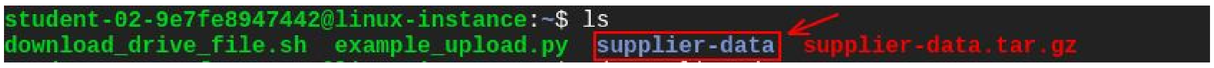


```
student-04-0c861ce837b1@linux-instance:~$ ./download_drive_file.sh 1LePo57dJcgzoK4uiI_48S01Etck7w_5f supplier-data.tar.gz
--2020-03-04 14:08:34--  https://docs.google.com/uc?export=download&confirm=gk_&id=1LePo57dJcgzoK4uiI_48S01Etck7w_5f
Resolving docs.google.com (docs.google.com)... 74.125.126.113, 74.125.126.100, 74.125.126.139, ...
Connecting to docs.google.com (docs.google.com)|74.125.126.113|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘supplier-data.tar.gz’

supplier-data.tar.gz                              [ <=>

2020-03-04 14:08:34 (41.4 MB/s) - ‘supplier-data.tar.gz’ saved [3271]

student-04-0c861ce837b1@linux-instance:~$ ls
download_drive_file.sh  example_upload.py  supplier-data.tar.gz
```

You have now downloaded a file named supplier-data.tar.gz containing the supplier's data. Let's extract the contents from this file using the following command:

```
tar xf ~/supplier-data.tar.gz
```

This creates a directory named supplier-data, that contains subdirectories named images and descriptions.

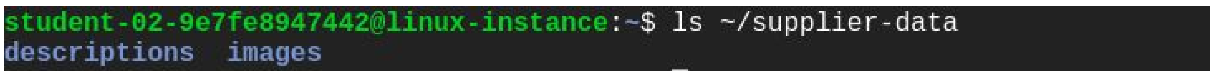


```
student-02-9e7fe8947442@linux-instance:~$ ls
download_drive_file.sh  example_upload.py  supplier-data  supplier-data.tar.gz
```

List contents of the supplier-data directory using the following command:

```
ls ~/supplier-data
```

Output:

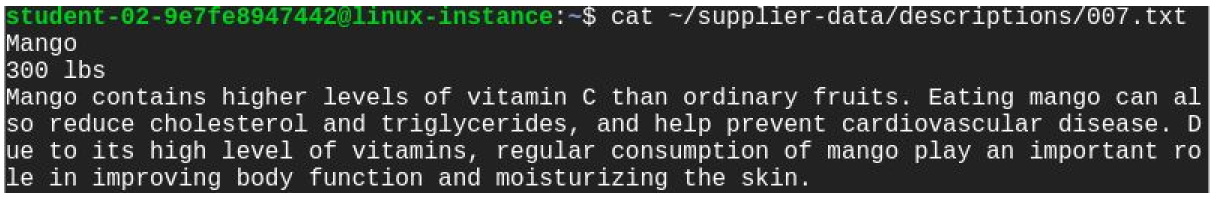


```
student-02-9e7fe8947442@linux-instance:~$ ls ~/supplier-data
descriptions  images
```

The subdirectory images contain images of various fruits, while the descriptions subdirectory has text files containing the description of each fruit. You can have a look at any of these text files using cat command.

```
cat ~/supplier-data/descriptions/007.txt
```

Output:



```
student-02-9e7fe8947442@linux-instance:~$ cat ~/supplier-data/descriptions/007.txt
Mango
300 lbs
Mango contains higher levels of vitamin C than ordinary fruits. Eating mango can al
so reduce cholesterol and triglycerides, and help prevent cardiovascular disease. D
ue to its high level of vitamins, regular consumption of mango play an important ro
le in improving body function and moisturizing the skin.
```

The first line contains the name of the fruit followed by the weight of the fruit and finally the description of the fruit.

Working with supplier images

In this section, you will write a Python script named changeImage.py to process the supplier images. You will be using the PIL library to update all images within ~/supplier-data/images directory to the following specifications:

- **Size**: Change image resolution from **3000x2000** to **600x400** pixel
- **Format**: Change image format from **.TIFF** to **.JPEG**

Create and open the file using nano editor.

```
nano ~/changeImage.py
```

Add a shebang line in the first line.

```
#!/usr/bin/env python3
```

This is the challenge section, where you will be writing a script that satisfies the above objectives.

Note: The raw images from images subdirectory contains alpha transparency layers. So, it is better to first convert RGBA 4-channel format to RGB 3-channel format before processing the images. Use convert("RGB") method for converting RGBA to RGB image.

After processing the images, save them in the same path ~/supplier-data/images, with a JPEG extension.

Once you have completed editing the changeImage.py script, save the file by clicking **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Grant executable permissions to the changeImage.py script.

```
sudo chmod +x ~/changeImage.py
```

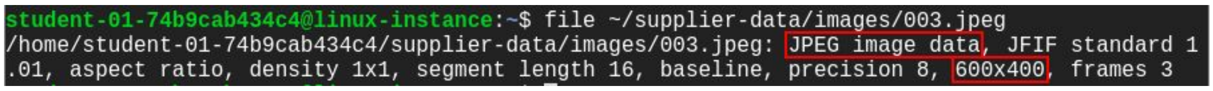
Now run the changeImage.py script:

```
./changeImage.py
```

Now, let's check the specifications of the images you just updated. Open any image using the following command:

```
file ~/supplier-data/images/003.jpeg
```

Output:



```
student-01-74b9cab434c4@linux-instance:~$ file ~/supplier-data/images/003.jpeg
/home/student-01-74b9cab434c4/supplier-data/images/003.jpeg: JPEG image data, JFIF standard 1
.01, aspect ratio, density 1x1, segment length 16, baseline, precision 8, 600x400, frames 3
```

Uploading images to web server

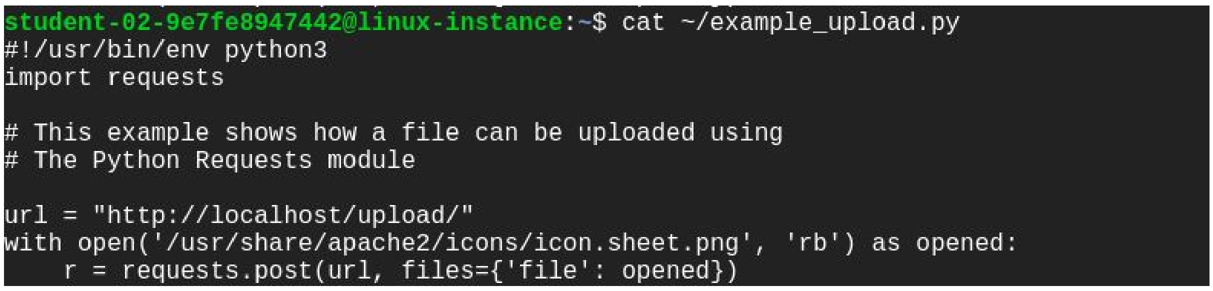
You have modified the fruit images through changeImage.py script. Now, you will have to upload these modified images to the web server that is handling the fruit catalog. To do that, you'll have to use the Python requests module to send the file contents to the [linux-instance-IP-Address]/upload URL.

Copy the external IP address of your instance from the Connection Details Panel on the left side and enter the IP address in a new web browser tab. This opens a web page displaying the text "Fruit Catalog".

In the home directory, you'll have a script named example_upload.py to upload images to the running fruit catalog web server. To view the example_upload.py script use the cat command.

```
cat ~/example_upload.py
```

Output:



```
student-02-9e7fe8947442@linux-instance:~$ cat ~/example_upload.py
#!/usr/bin/env python3
import requests

# This example shows how a file can be uploaded using
# The Python Requests module

url = "http://localhost/upload/"
with open('/usr/share/apache2/icons/icon.sheet.png', 'rb') as opened:
    r = requests.post(url, files={'file': opened})
```

In this script, we are going to upload a sample image named icon.sheet.png.

Grant executable permission to the example_upload.py script.

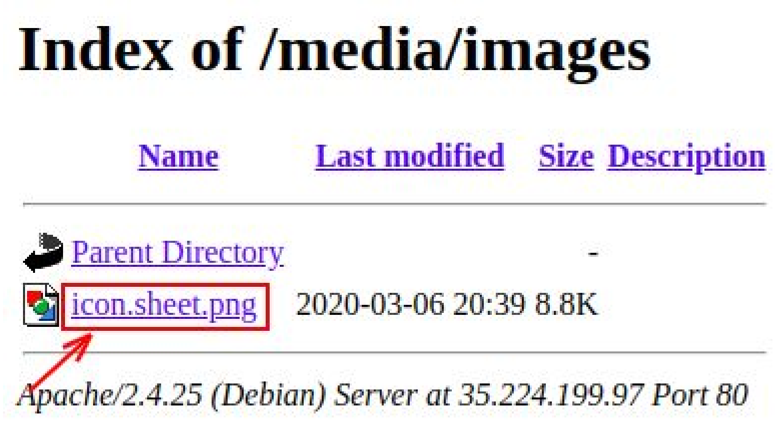
```
sudo chmod +x ~/example_upload.py
```

Execute the example_upload.py script, which will upload the images.

```
./example_upload.py
```

Now check out that the file icon.sheet.png was uploaded to the web server by visiting the URL [linux-instance-IP-Address]/media/images/, followed by clicking on the file name.

Output:



Index of /media/images

```
        Name          Last modified    Size  Description
Parent Directory                         -
icon.sheet.png     2020-03-06 20:39  8.8K
Apache/2.4.25 (Debian) Server at 35.224.199.97 Port 80
```

In a similar way, you are going to write a script named supplier_image_upload.py that takes the **jpeg** images from the supplier-data/images directory that you've processed previously and uploads them to the web server fruit catalog.

Use the nano editor to create a file named supplier_image_upload.py:

```
nano ~/supplier_image_upload.py
```

Complete the script with the same technique as used in the file example_upload.py.

Once you have completed editing the supplier_image_upload.py script, save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Grant executable permission to the changeImage.py script.

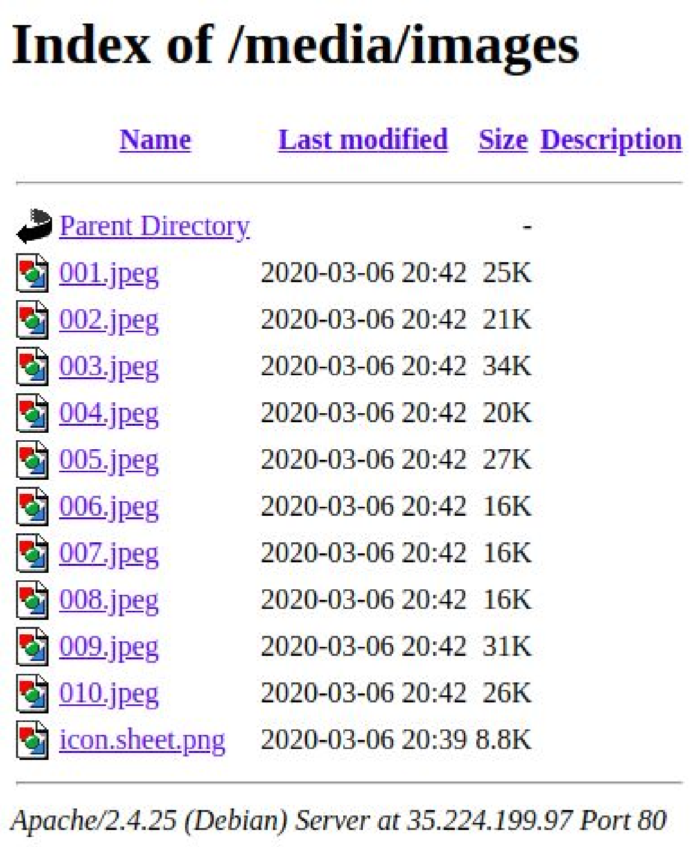
```
sudo chmod +x ~/supplier_image_upload.py
```

Run the changeImage.py script.

```
./supplier_image_upload.py
```

Refresh the URL opened earlier, and now you should find all the images uploaded successfully.

Output:

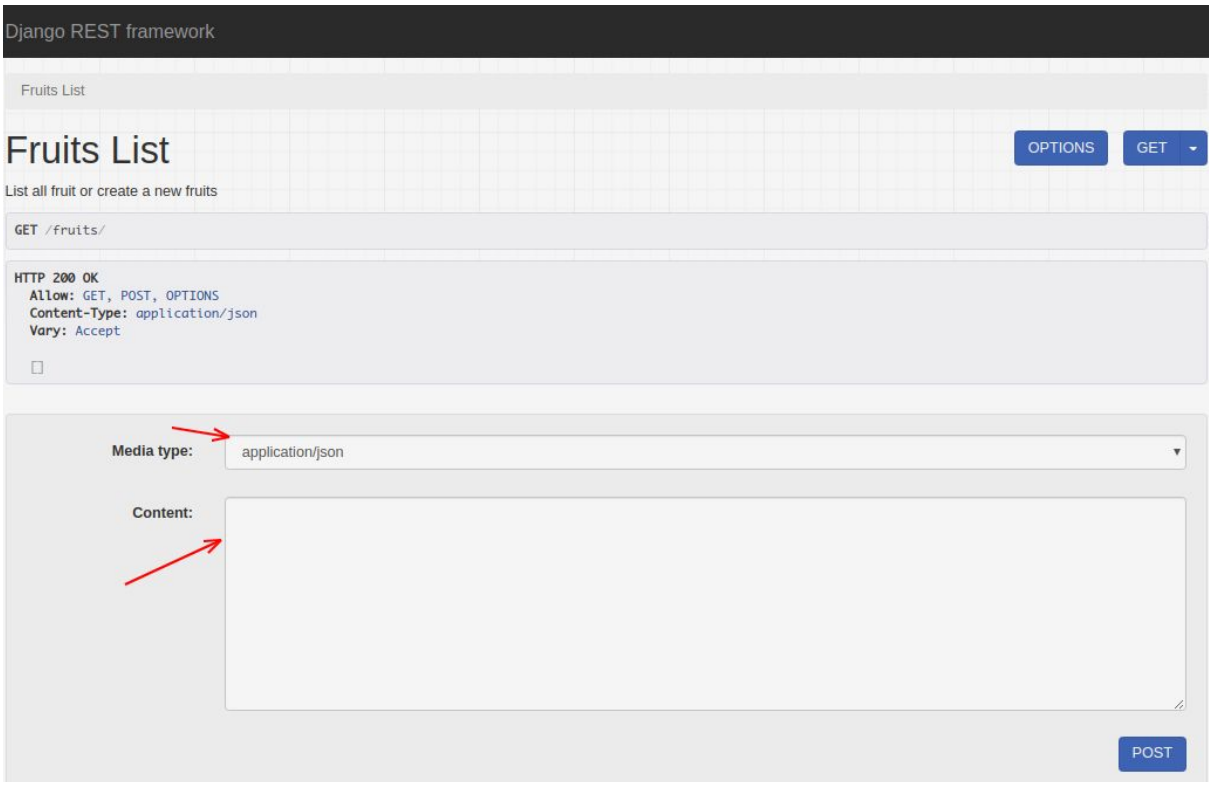


```
Index of /media/images

      Name            Last modified      Size  Description
--------------------------------------------------------------
  Parent Directory                          -
  001.jpeg          2020-03-06 20:42  25K
  002.jpeg          2020-03-06 20:42  21K
  003.jpeg          2020-03-06 20:42  34K
  004.jpeg          2020-03-06 20:42  20K
  005.jpeg          2020-03-06 20:42  27K
  006.jpeg          2020-03-06 20:42  16K
  007.jpeg          2020-03-06 20:42  16K
  008.jpeg          2020-03-06 20:42  16K
  009.jpeg          2020-03-06 20:42  31K
  010.jpeg          2020-03-06 20:42  26K
  icon.sheet.png    2020-03-06 20:39 8.8K
--------------------------------------------------------------
Apache/2.4.25 (Debian) Server at 35.224.199.97 Port 80
```

Uploading the descriptions

The Django server is already set up to show the fruit catalog for your company. You can visit the main website by entering linux-instance-IP-Address in the URL bar or by removing /media/images from the existing URL opened earlier. The interface looks like this:

Check out the Django REST framework, by navigating to linux-instance-IP-Address/fruits in your browser.


Django REST framework

Fruits List

Fruits List

List all fruit or create a new fruits

```
GET /fruits/
```

```
HTTP 200 OK
Allow: GET, POST, OPTIONS
Content-Type: application/json
Vary: Accept
```

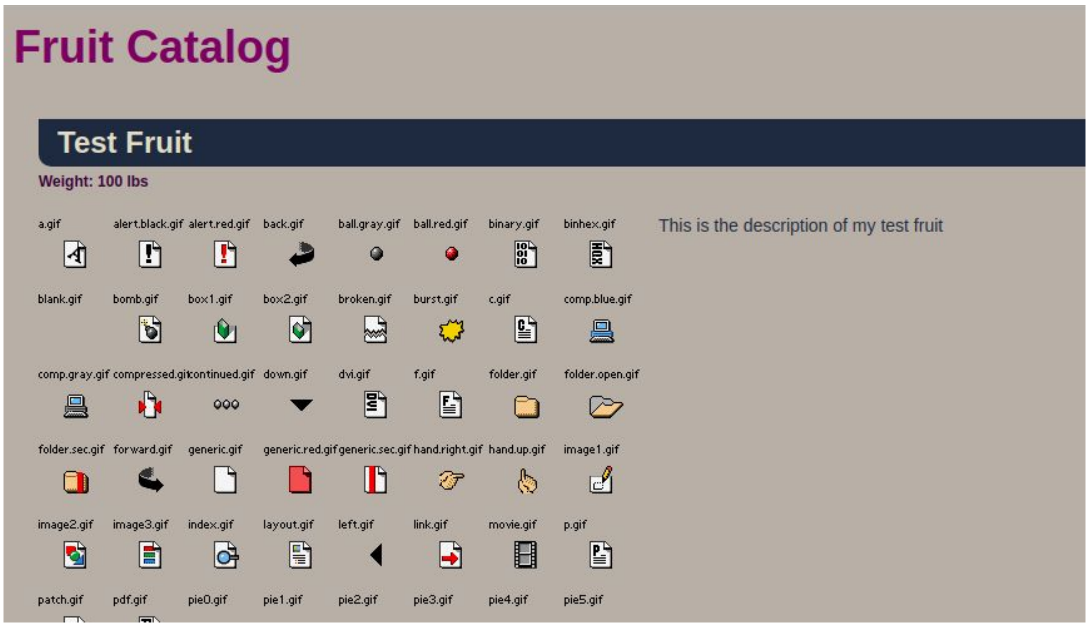
Media type: application/json

Content:

Currently, there are no products in the fruit catalog web-server. You can create a test fruit entry by entering the following into the **content** field:

```
{"name": "Test Fruit", "weight": 100, "description": "This is the description of my test fruit", "image_name": "icon.sheet.png"}
```

After entering the above data into the content field click on the POST button. Now visit the main page of your website (by going to http://[linux-instance-external-IP]/), and the new test fruit you uploaded appears.


Fruit Catalog

Test Fruit

Weight: 100 lbs

This is the description of my test fruit

To add fruit images and their descriptions from the supplier on the fruit catalog web-server, create a new Python script that will automatically POST the **fruit images** and their respective **description** in JSON format.

Write a Python script named run.py to process the text files (001.txt, 003.txt ...) from the supplier-data/descriptions directory. The script should turn the data into a JSON dictionary by adding all the required fields, including the image associated with the fruit (image_name), and uploading it to http://[linux-instance-external-IP]/fruits using the Python **requests** library.

Create run.py using the nano editor:

```
nano ~/run.py
```

Add the shebang line and import necessary libraries.

```
#! /usr/bin/env python3

import os

import requests
```

Now, you'll have to process the .txt files (named 001.txt, 002.txt, ...) in the supplier-data/descriptions/ directory and save them in a data structure so that you can then upload them via JSON. Note that all files are written in the following format, with each piece of information on its own line:

- name
- weight (in lbs)
- description

The data model in the Django application fruit has the following fields: name, weight, description and image_name. The weight field is defined as an **integer** field. So when you process the weight information of the fruit from the .txt file, you need to convert it into an integer. For example if the weight is "**500 lbs**", you need to **drop "lbs"** and **convert "500" to an integer**.

The image_name field will allow the system to find the image associated with the fruit. Don't forget to add all fields, including the image_name! The final JSON object should be similar to:

```
{"name": "Watermelon", "weight": 500, "description": "Watermelon is good for relieving heat, eliminating annoyance and quenching thirst. It contains a lot of water, which is good for relieving the symptoms of acute fever immediately. The sugar and salt contained in watermelon can diuretic and eliminate kidney inflammation. Watermelon also contains substances that can lower blood pressure.", "image_name": "010.jpeg"}
```

Iterate over all the fruits and use **post** method from Python requests library to upload all the data to the URL http://[linux-instance-external-IP]/fruits

Once you complete editing run.py script, save the file by clicking **Ctrl-o**, **Enter** key, and **Ctrl-x**.

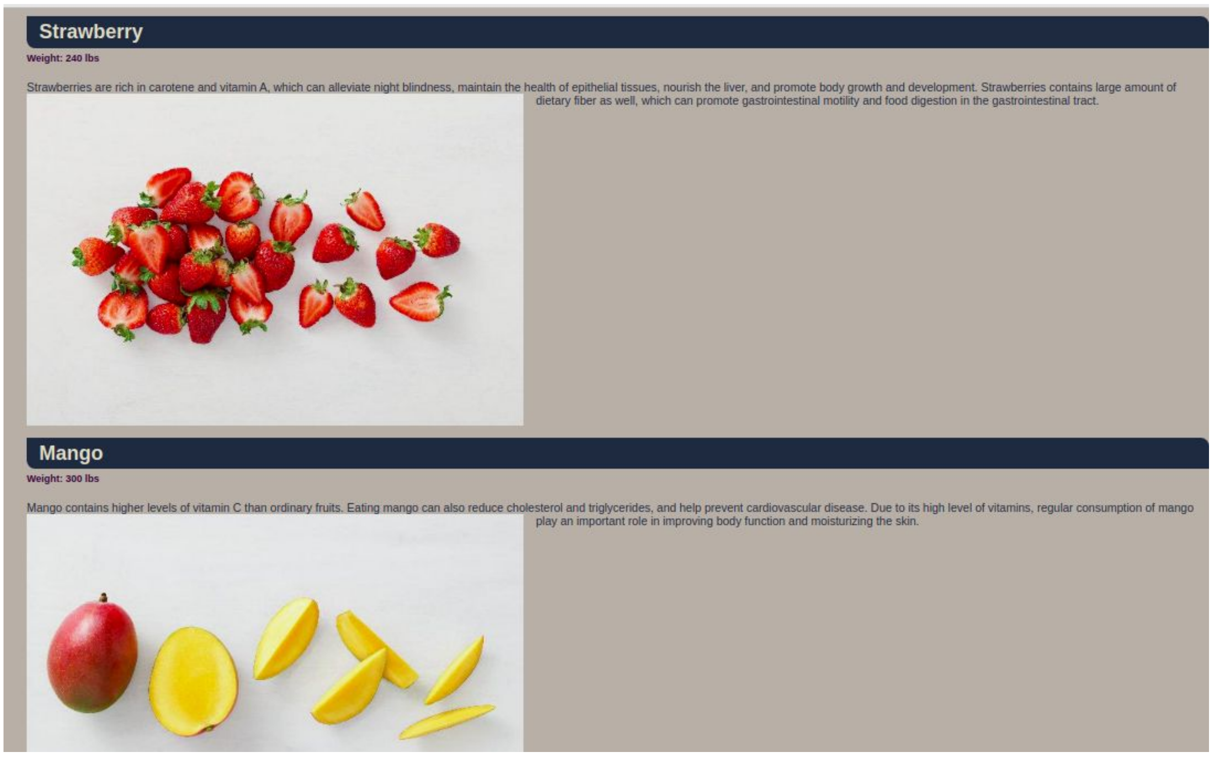
Grant executable permission to the run.py script.

```
sudo chmod +x ~/run.py
```

Run the run.py script:

```
./run.py
```

Now go to the main page of your website (by going to http://[linux-instance-IP-Address]/) and check out how the new fruits appear.



Strawberry

Weight: 240 lbs

Strawberries are rich in carotene and vitamin A, which can alleviate night blindness, maintain the health of epithelial tissues, nourish the liver, and promote body growth and development. Strawberries contains large amount of dietary fiber as well, which can promote gastrointestinal motility and food digestion in the gastrointestinal tract.

Mango

Weight: 300 lbs

Mango contains higher levels of vitamin C than ordinary fruits. Eating mango can also reduce cholesterol and triglycerides, and help prevent cardiovascular disease. Due to its high level of vitamins, regular consumption of mango play an important role in improving body function and moisturizing the skin.

Generate a PDF report and send it through email

Once the images and descriptions have been uploaded to the fruit store web-server, you will have to generate a PDF file to send to the supplier, indicating that the data was correctly processed. To generate PDF reports, you can use the ReportLab library. The content of the report should look like this:

```
Processed Update on <Today's date>

[blank line]

name: Apple

weight: 500 lbs

[blank line]
```

```
name: Avocado

weight: 200 lbs

[blank line]

...
```

Script to generate a PDF report

Create a script reports.py to generate PDF report to supplier using the nano editor:

```
nano ~/reports.py
```

Add a shebang line in the first line.

```
#!/usr/bin/env python3
```

Using the reportlab Python library, define the method generate_report to build the PDF reports. We have already covered how to generate PDF reports in an earlier lesson; you will want to use similar concepts to create a PDF report named **processed.pdf**.

Once you have finished editing the script reports.py, save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Create another script named report_email.py to process supplier fruit description data from supplier-data/descriptions directory. Use the following command to create report_email.py.

```
nano ~/report_email.py
```

Add a shebang line.

```
#!/usr/bin/env python3
```

Import all the necessary libraries(os, datetime and reports) that will be used to process the text data from the supplier-data/descriptions directory into the format below:

```
name: Apple

weight: 500 lbs

[blank line]

name: Avocado

weight: 200 lbs

[blank line]

...
```

Once you have completed this, call the main method which will process the data and call the generate_report method from the reports module:

```
if __name__ == "__main__":
```

You will need to pass the following arguments to the reports.generate_report method: the text description processed from the text files as the paragraph argument, the report title as the title argument, and the file path of the PDF to be generated as the attachment argument (use '/tmp/processed.pdf')

```
 reports.generate_report(attachment, title, paragraph)
```

Once you have completed the report_email.py script. Save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Send report through email

Once the PDF is generated, you need to send the email using the emails.generate_email() and emails.send_email() methods.

Create emails.py using the nano editor using the following command:

```
nano ~/emails.py
```

Define generate_email and send_email methods by importing necessary libraries.

Once you have finished editing the emails.py script, save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Now, open the report_email.py script using the nano editor:

```
nano ~/report_email.py
```

Once you define the generate_email and send_email methods, call the methods under the main method after creating the PDF report:

```
if __name__ == "__main__":
```

Use the following details to pass the parameters to emails.generate_email():

- **From:** automation@example.com
- **To:** username@example.com
  - Replace username with the username given in the Connection Details Panel on the right hand side.
- **Subject line**: Upload Completed - Online Fruit Store
- **E-mail Body:** All fruits are uploaded to our website successfully. A detailed list is attached to this email.
- **Attachment:** Attach the path to the file processed.pdf

Once you have finished editing the report_email.py script, save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Grant executable permissions to the script report_email.py.

```
sudo chmod +x ~/report_email.py
```

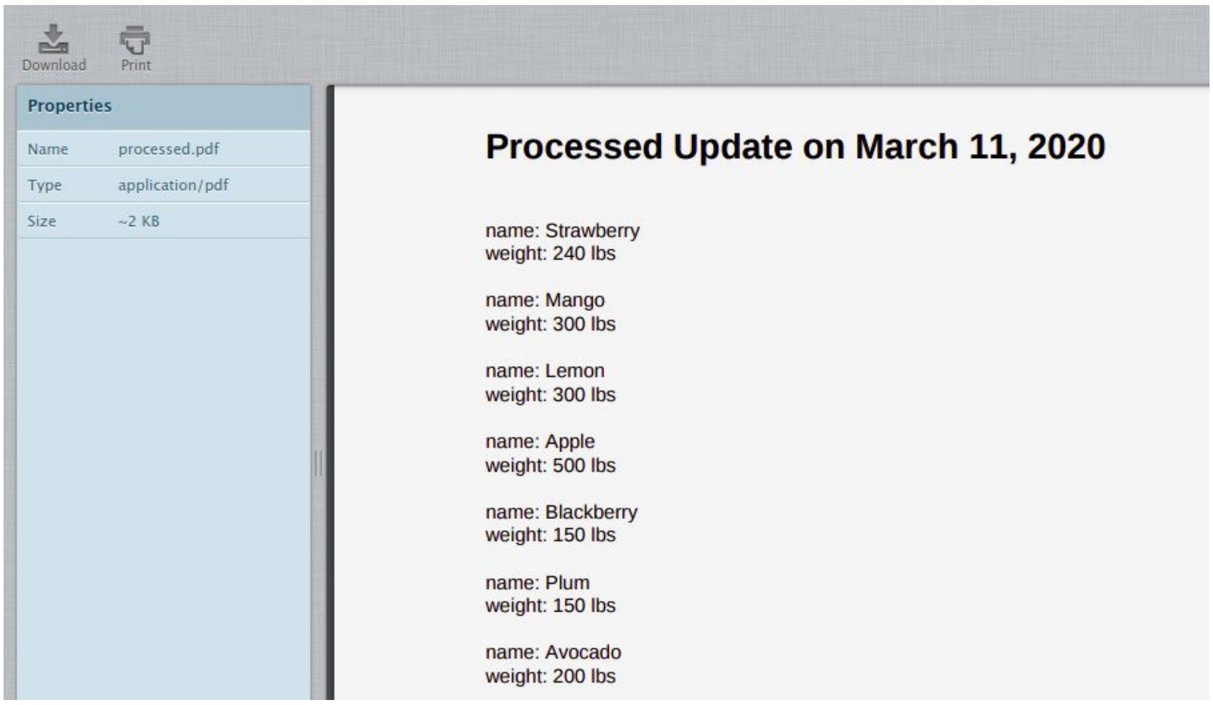
Run the report_email.py script.

```
./report_email.py
```

Now, check the webmail by visiting [linux-instance-external-IP]/webmail. Here, you'll need a login to **roundcube** using the username and password mentioned in the Connection Details Panel on the left hand side, followed by clicking **Login**.

Now you should be able to see your inbox, with one unread email. Open the mail by double clicking on it. There should be a report in PDF format attached to the mail. View the report by opening it.

Output:



Name	processed.pdf
Type	application/pdf
Size	~2 KB

Processed Update on March 11, 2020

```
name: Strawberry
weight: 240 lbs

name: Mango
weight: 300 lbs

name: Lemon
weight: 300 lbs

name: Apple
weight: 500 lbs

name: Blackberry
weight: 150 lbs

name: Plum
weight: 150 lbs

name: Avocado
weight: 200 lbs
```

Health check

This is the last part of the lab, where you will have to write a Python script named health_check.py that will run in the background monitoring some of your system statistics: CPU usage, disk space, available memory and name resolution. Moreover, this Python script should send an email if there are problems, such as:

- Report an error if CPU usage is over 80%
- Report an error if available disk space is lower than 20%

- Report an error if available memory is less than 500MB
- Report an error if the hostname "localhost" cannot be resolved to "127.0.0.1"

Create a python script named health_check.py using the nano editor:

```
nano ~/health_check.py
```

Add a shebang line.

```
#!/usr/bin/env python3
```

Import the necessary Python libraries (eg. shutil, psutil) to write this script.

Complete the script to check the system statistics every 60 seconds, and in event of any issues detected among the ones mentioned above, an email should be sent with the following content:

- **From:** automation@example.com
- **To:** username@example.com
    - Replace username with the username given in the Connection Details Panel on the right hand side.
- **Subject line**:

Case	**Subject line**
CPU usage is over 80%	Error - CPU usage is over 80%
Available disk space is lower than 20%	Error - Available disk space is less than 20%
available memory is less than 500MB	Error - Available memory is less than 500MB
hostname "localhost" cannot be resolved to "127.0.0.1"	Error - localhost cannot be resolved to 127.0.0.1

- **E-mail Body:** Please check your system and resolve the issue as soon as possible.

Note: There is no attachment file here, so you must be careful while defining the generate_email() method in the emails.py script or you can create a separate generate_error_report() method for handling non-attachment email.

Once you have completed the health_check.py script. Save the file by typing **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Grant executable permissions to the script health_check.py.

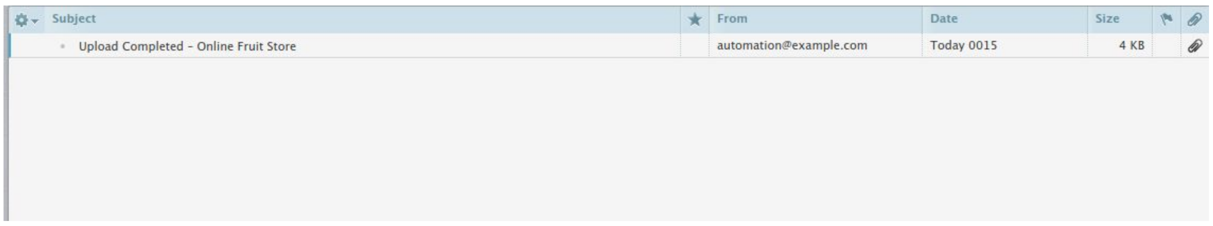
```
sudo chmod +x ~/health_check.py
```

Run the file.

```
./health_check.py
```

Next, go to the webmail inbox and refresh it. There should only be an email something goes wrong, so hopefully you don't see a new email.

Output:



Subject	★	From	Date	Size		
Upload Completed - Online Fruit Store		automation@example.com	Today 0015	4 KB		

To test out your script, you can install the stress tool.

```
sudo apt install stress
```

Next, call the tool using a good number of CPUs to fully load our CPU resources:

```
stress --cpu 8
```

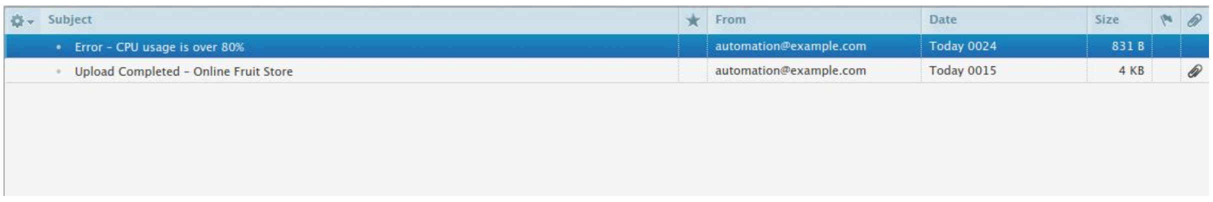
Allow the stress test to run, as it will maximize our CPU utilization. Now run health_check.py by opening another SSH connection to the linux-instance. Navigate to Accessing the virtual machine on the navigation pane on the right-hand side to open another connection to the instance.

Now run the script:

```
./health_check.py
```

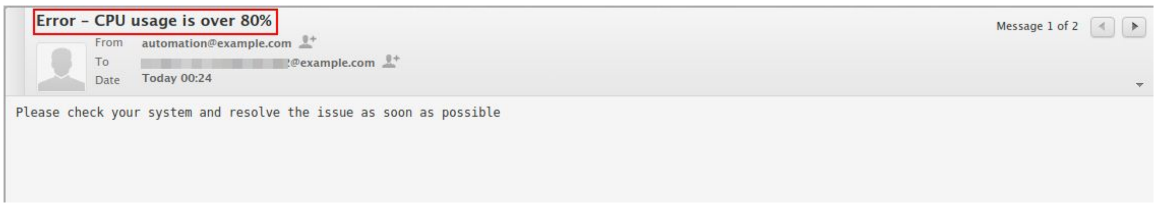
Check your inbox for any new email.

Output:



Subject	★	From	Date	Size		
Error - CPU usage is over 80%		automation@example.com	Today 0024	831 B		
Upload Completed - Online Fruit Store		automation@example.com	Today 0015	4 KB		

Open the email with the subject "Error - CPU usage is over 80%" by double clicking it.



Error - CPU usage is over 80%

From automation@example.com

To [redacted]@example.com

Date Today 00:24

Message 1 of 2

```
Please check your system and resolve the issue as soon as possible
```

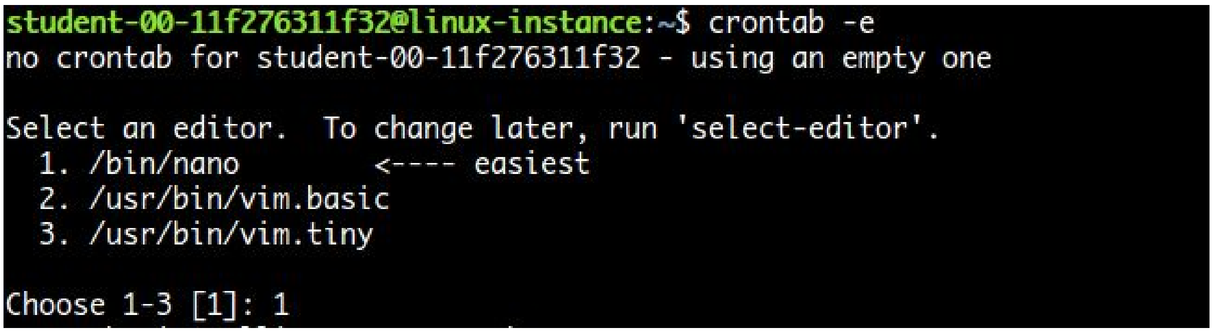
Close the stress --cpu command by clicking **Ctrl-c**.

Now, you will be setting a cron job that executes the script health_check.py every 60 seconds and sends health status to the respective user.

To set a user cron job use the following command:

```
crontab -e
```

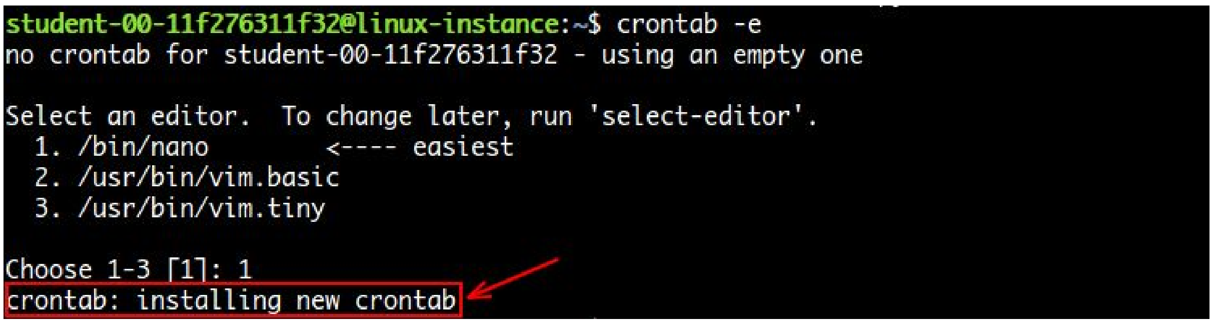
Output:



```
student-00-11f276311f32@linux-instance:~$ crontab -e
no crontab for student-00-11f276311f32 - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.basic
  3. /usr/bin/vim.tiny

Choose 1-3 [1]: 1
```

Enter 1 to open in the nano editor. Now, set the complete path for health_check.py script, and save by clicking **Ctrl-o**, **Enter** key, and **Ctrl-x**.

Output:



```
student-00-11f276311f32@linux-instance:~$ crontab -e
no crontab for student-00-11f276311f32 - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.basic
  3. /usr/bin/vim.tiny

Choose 1-3 [1]: 1
crontab: installing new crontab
```