Automate updating catalog information

2 hoursFree

**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.

Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

**Note:** For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

Start Lab

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:



**Accessing the virtual machine**

Please find one of the three relevant options below based on your device's operating system.

**Note:** Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you’ll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM’s External IP address to connect.

**Download your PPK key file**

You can download the VM’s private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.



**Connect to your VM using SSH and PuTTY**

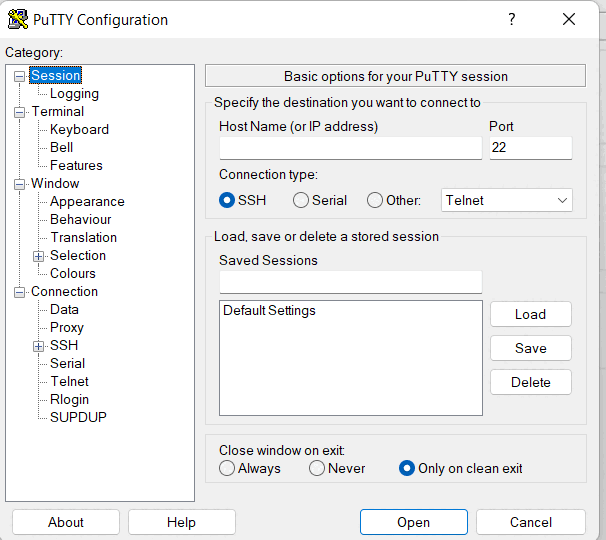
1. You can download Putty from [here](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)
2. In the **Host Name (or IP address)** box, enter username@external\_ip\_address.

**Note:** Replace **username** and **external\_ip\_address** with values provided in the lab.



1. In the **Connection** list, expand **SSH**.
2. Then expand **Auth** by clicking on **+** icon.
3. Now, select the **Credentials** from the **Auth** list.
4. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
5. Click on the **Open** button.

**Note:** PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



1. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

**Common issues**

If PuTTY fails to connect to your Linux VM, verify that:

* You entered **<username>**@**<external ip address>** in PuTTY.
* You downloaded the fresh new PPK file for this lab from Qwiklabs.
* You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to the VM using the local Terminal application**

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
   * To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
   * To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

**Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

chmod 600 ~/Downloads/qwikLABS-XXXXX.pem

Copied!

content\_copy

ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address

Copied!

content\_copy



Option 3: Chrome OS users: Connecting to your VM via SSH

**Note:** Make sure you are not in **Incognito/Private mode** while launching the application.

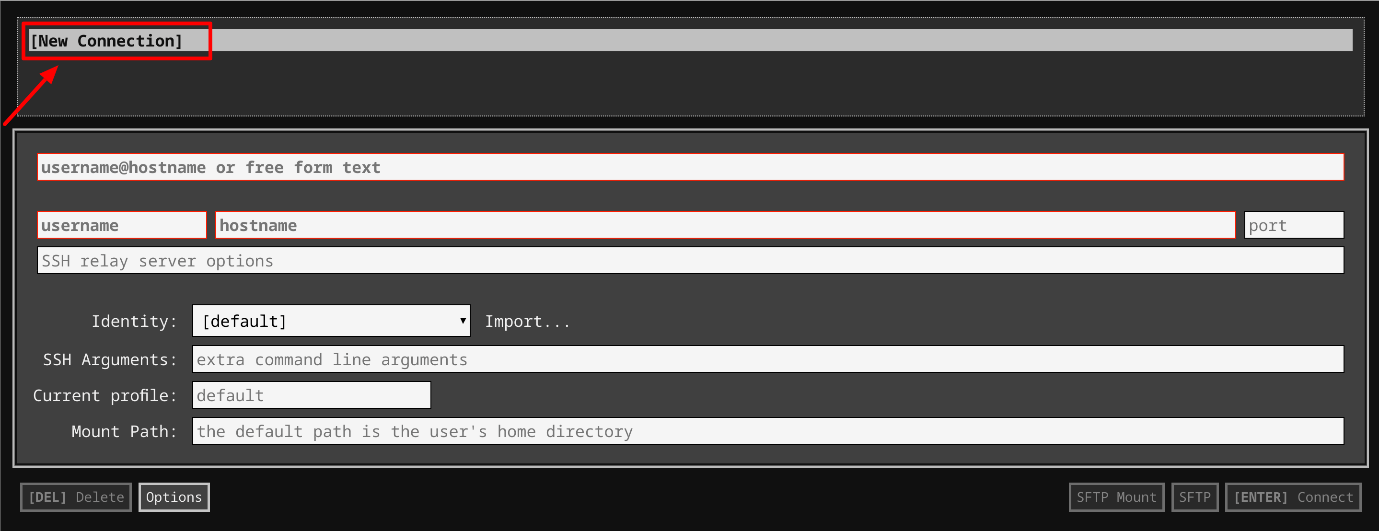
**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to your VM**

1. Add Secure Shell from [here](https://chrome.google.com/webstore/detail/secure-shell-app/pnhechapfaindjhompbnflcldabbghjo) to your Chrome browser.
2. Open the Secure Shell app and click on **[New Connection]**.



1. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.



1. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

**Note:** If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

1. Once your key is uploaded, click on the **[ENTER] Connect** button below.



1. For any prompts, type **yes** to continue.
2. You have now successfully connected to your Linux VM.

You're now ready to continue with the 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 ~/

Copied!

content\_copy

Output:



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

Copied!

content\_copy

Run the download\_drive\_file.sh shell script with the following arguments:

./download\_drive\_file.sh 1LePo57dJcgzoK4uiI\_48S01Etck7w\_5f supplier-data.tar.gz

Copied!

content\_copy

Output:

A screenshot of a computer

Description automatically generated with medium confidence

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

Copied!

content\_copy

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



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

ls ~/supplier-data

Copied!

content\_copy

Output:



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

Copied!

content\_copy

Output:

A screen shot of a computer

Description automatically generated with medium confidence

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

Copied!

content\_copy

Add a shebang line in the first line.

#!/usr/bin/env python3

Copied!

content\_copy

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

Copied!

content\_copy

Now run the changeImage.py script:

./changeImage.py

Copied!

content\_copy

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

Copied!

content\_copy

Output:



Click *Check my progress* to verify the objective.

Update image specifications

Check my progress

**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

Copied!

content\_copy

Output:

A screen shot of a computer

Description automatically generated with medium confidence

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

Copied!

content\_copy

Execute the example\_upload.py script, which will upload the images.

./example\_upload.py

Copied!

content\_copy

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:

A screenshot of a computer

Description automatically generated with low confidence

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

Copied!

content\_copy

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 supplier\_image\_upload.py script.

sudo chmod +x ~/supplier\_image\_upload.py

Copied!

content\_copy

Run the supplier\_image\_upload.py script.

./supplier\_image\_upload.py

Copied!

content\_copy

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

Output:

A screenshot of a computer

Description automatically generated with medium confidence

Click *Check my progress* to verify the objective.

Upload images to the web server

Check my progress

**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:

A picture containing text, font, screenshot, design

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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

Copied!

content\_copy

Add the shebang line and import necessary libraries.

#! /usr/bin/env python3

Copied!

content\_copy

import os

import requests

Copied!

content\_copy

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

Copied!

content\_copy

Run the run.py script:

./run.py

Copied!

content\_copy

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.

A picture containing fruit, text, screenshot, natural foods

Description automatically generated

Click *Check my progress* to verify the objective.

Upload the descriptions

Check my progress

**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

Copied!

content\_copy

Add a shebang line in the first line.

#!/usr/bin/env python3

Copied!

content\_copy

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

Copied!

content\_copy

Add a shebang line.

#!/usr/bin/env python3

Copied!

content\_copy

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\_\_":

Copied!

content\_copy

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)

Copied!

content\_copy

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

Copied!

content\_copy

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

Copied!

content\_copy

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\_\_":

Copied!

content\_copy

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

Copied!

content\_copy

Run the report\_email.py script.

./report\_email.py

Copied!

content\_copy

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:

A screenshot of a computer

Description automatically generated with medium confidence

Click *Check my progress* to verify the objective.

Generate PDF and send through email

Check my progress

**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

Copied!

content\_copy

Add a shebang line.

#!/usr/bin/env python3

Copied!

content\_copy

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

Copied!

content\_copy

Run the file.

./health\_check.py

Copied!

content\_copy

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:

A picture containing text, screenshot

Description automatically generated

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

sudo apt install stress

Copied!

content\_copy

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

stress --cpu 8

Copied!

content\_copy

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

Copied!

content\_copy

Check your inbox for any new email.

Output:

A screenshot of a computer

Description automatically generated with medium confidence

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

A screenshot of a computer

Description automatically generated with medium confidence

Click *Check my progress* to verify the objective.

Health check

Check my progress

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

Copied!

content\_copy

Output:

A screen shot of a computer

Description automatically generated with medium confidence

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:

A screenshot of a computer program

Description automatically generated with medium confidence

**Congratulations!**

Congrats! You've successfully created a python script that processes images and descriptions and then updates your company's online website to add the new products. You have also generated a PDF report and sent it by email. Finally, you have also set up monitoring of the system's health.

**End your lab**

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you’ve used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

* 1 star = Very dissatisfied
* 2 stars = Dissatisfied
* 3 stars = Neutral
* 4 stars = Satisfied
* 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.