|  |
| --- |
| # import python modules |
|  | import pika |
|  | import sys |
|  | import webbrowser |
|  | import csv |
|  | import time |
|  |  |
|  | ######################################################## |
|  |  |
|  | # define variables/constants/options |
|  | host = "localhost" |
|  | csv\_file = "smoker-temps.csv" |
|  | smoker\_queue = "01-smoker" |
|  | foodA\_queue = "02-food-A" |
|  | foodB\_queue = "03-food-B" |
|  | show\_offer = True # (RabbitMQ Server option - T=on, F=off) |
|  |  |
|  | ######################################################## |
|  |  |
|  | # define functions |
|  | ## define option to open RabbitMQ admin webpage |
|  | def offer\_rabbitmq\_admin\_site(show\_offer): |
|  | # includes show\_offer variable - option to turn off the offer later in the code |
|  | if show\_offer == True: |
|  | """Offer to open the RabbitMQ Admin website""" |
|  | ans = input("Would you like to monitor RabbitMQ queues? y or n ") |
|  | print() |
|  | if ans.lower() == "y": |
|  | webbrowser.open\_new("http://localhost:15672/#/queues") |
|  | print() |
|  |  |
|  | ## define delete\_queue |
|  | def delete\_queue(host: str, queue\_name: str): |
|  | """ |
|  | Delete queues each time we run the program to clear out old messages. |
|  | """ |
|  | conn = pika.BlockingConnection(pika.ConnectionParameters(host)) |
|  | ch = conn.channel() |
|  | ch.queue\_delete(queue=queue\_name) |
|  |  |
|  | ## define a message to send to queue |
|  | def publish\_message\_to\_queue(host: str, queue\_name: str, message: str): |
|  | """ |
|  | Creates and sends a message to the queue each execution. |
|  | This process runs and finishes. |
|  |  |
|  | Parameters: |
|  | host (str): the host name or IP address of the RabbitMQ server |
|  | queue\_name (str): the name of the queue |
|  | message (str): the message to be sent to the queue |
|  | """ |
|  | ### Get a connection to RabbitMQ and create a channel |
|  | try: |
|  | # create a connection to the RabbitMQ server |
|  | conn = pika.BlockingConnection(pika.ConnectionParameters(host)) |
|  | # use the connection to create a communication channel |
|  | ch = conn.channel() |
|  | # declare a durable queue (will survive a RabbitMQ server restart |
|  | # and help ensure messages are processed in order) |
|  | # messages will not be deleted until the consumer acknowledges |
|  | ch.queue\_declare(queue=queue\_name, durable=True) |
|  | # use the channel to publish a message to the queue; each message passes through an exchange |
|  | ch.basic\_publish(exchange="", routing\_key=queue\_name, body=message) |
|  | # print a message to the console for the user |
|  | print(f" [x] Sent {message} to {queue\_name}") |
|  | except pika.exceptions.AMQPConnectionError as e: |
|  | print(f"Error: Connection to RabbitMQ server failed: {e}") |
|  | sys.exit(1) |
|  | finally: |
|  | # close the connection to the server |
|  | conn.close() |
|  |  |
|  | # define getting/reading a message from the csv file & publishing to the queue |
|  | def get\_message\_from\_csv(input\_file): |
|  | """ |
|  | Read from csv input file. Send each row as a message to the queue. |
|  | """ |
|  |  |
|  | # read from a csv file |
|  | input\_file = open(csv\_file, "r") |
|  | reader = csv.reader(input\_file, delimiter=',') |
|  |  |
|  | # Skip reading the header row of csv |
|  | next(reader) |
|  |  |
|  | for row in reader: |
|  | # define the input strings that we want to convert into float data types |
|  | input\_string\_row1 = row[1] |
|  | input\_string\_row2 = row[2] |
|  | input\_string\_row3 = row[3] |
|  |  |
|  | # remove blank/empty strings and replace them with zeroes |
|  | to\_convert\_column1 = input\_string\_row1.replace('', '0') |
|  | to\_convert\_column2 = input\_string\_row2.replace('', '0') |
|  | to\_convert\_column3 = input\_string\_row3.replace('', '0') |
|  |  |
|  | # Convert strings (now with 0s instead of empty strings) to float types |
|  | float\_row1 = float(to\_convert\_column1) |
|  | float\_row2 = float(to\_convert\_column2) |
|  | float\_row3 = float(to\_convert\_column3) |
|  |  |
|  | # turn column values into fstrings |
|  | fstring\_time = f"{row[0]}" |
|  | fstring\_channel1 = f"{row[1]}" |
|  | fstring\_channel2 = f"{row[2]}" |
|  | fstring\_channel3 = f"{row[3]}" |
|  |  |
|  | # use an fstring to create messages from our data |
|  | fstring\_message\_smoker = f"[{fstring\_time}, {fstring\_channel1}]" |
|  | fstring\_message\_foodA = f"[{fstring\_time}, {fstring\_channel2}]" |
|  | fstring\_message\_foodB = f"[{fstring\_time}, {fstring\_channel3}]" |
|  |  |
|  | # prepare a binary (1s and 0s) message to stream |
|  | # be careful: these are case sensitive! |
|  | message\_smoker = fstring\_message\_smoker.encode() |
|  | message\_foodA = fstring\_message\_foodA.encode() |
|  | message\_foodB = fstring\_message\_foodB.encode() |
|  |  |
|  | # publish to queues using routing |
|  | if float\_row1 > 0: publish\_message\_to\_queue(host, smoker\_queue, message\_smoker) |
|  | if float\_row2 > 0: publish\_message\_to\_queue(host, foodA\_queue, message\_foodA) |
|  | if float\_row3 > 0: publish\_message\_to\_queue(host, foodB\_queue, message\_foodB) |
|  | else: pass # print() |
|  |  |
|  | # slowly read a row half minute (30 seconds) |
|  | # can change this to 1 second for testing purposes - makes it go faster |
|  | time.sleep(30) |
|  |  |
|  | ######################################################## |
|  |  |
|  | # Run program |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | # if show\_offer = True, ask the user if they'd like to open the RabbitMQ Admin site |
|  | offer\_rabbitmq\_admin\_site(show\_offer) |
|  | # delete queues to clear old messages |
|  | delete\_queue(host, smoker\_queue) |
|  | delete\_queue(host, foodA\_queue) |
|  | delete\_queue(host, foodB\_queue) |
|  | # get the message from the csv input file and send to queue |
|  | get\_message\_from\_csv(csv\_file) |