|  |
| --- |
| # import python modules |
|  | import pika |
|  | import sys |
|  | import time |
|  | from collections import deque |
|  |  |
|  | ######################################################## |
|  |  |
|  | # 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) |
|  |  |
|  | # set alert limits |
|  | smoker\_alert\_limit = 15 # temp decrease of this amount sends a smoker alert |
|  | food\_stall\_alert\_limit = 1 # temp change of this amount sends a food stall alert |
|  |  |
|  | # Time windowing - Create deques to store that last n messages |
|  | smoker\_deque = deque(maxlen=5) # limited to 5 items (the 5 most recent readings) |
|  | foodA\_deque = deque(maxlen=20) # limited to 20 items (the 20 most recent readings) |
|  | foodB\_deque = deque(maxlen=20) # limited to 20 items (the 20 most recent readings) |
|  |  |
|  | ######################################################## |
|  |  |
|  | # define callback functions (called when message is received - 1 per queue) |
|  |  |
|  | def smoker\_callback(ch, method, properties, body): |
|  | """ Define behavior on getting a message in the 01-smoker queue. |
|  | Monitor smoker temperature. Send an alert if the smoker temp decreases by 15 F or more in 2.5 min (or 5 readings). """ |
|  |  |
|  | # receive & decode the binary message body to a string |
|  | print(f" [x] Received {body.decode()} on 01-smoker") |
|  | # simulate work |
|  | time.sleep(5) |
|  | # basic\_ack - acknowledge the message was received and processed (now it can be deleted from the queue) |
|  | ch.basic\_ack(delivery\_tag=method.delivery\_tag) |
|  |  |
|  | # create smoker deque to store x amount of recent messages |
|  | # add new message to our smoker deque |
|  | smoker\_deque.append(body.decode()) |
|  | # first item in deque (oldest) |
|  | smoker\_deque\_item1 = smoker\_deque[0] |
|  | # split the oldest message in the deque by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our date and timestamp from 5 messages ago (2.5 mins ago) |
|  | # the second list item [1] is the temp from 5 messages ago (2.5 mins ago) |
|  | smoker\_deque\_item1\_split = smoker\_deque\_item1.split(", ") |
|  | # change to float and remove last 2 characters from string, which is the ']' character of our message |
|  | smoker\_deque\_item1\_temp = float(smoker\_deque\_item1\_split[1][:-1]) |
|  |  |
|  | # smoker current temp/current message code |
|  | smoker\_current\_timetemp = body.decode() |
|  | # split the current message by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our current date and timestamp being read in this message |
|  | # the second list item [1] is our current temp being read in this message |
|  | smoker\_current\_timetemp\_split = smoker\_current\_timetemp.split(", ") |
|  | # change temp to float and remove last 1 characters from string, which is the ']' character of our message |
|  | smoker\_current\_temp = float(smoker\_current\_timetemp\_split[1][:-1]) |
|  |  |
|  | # calculate smoker temperature change and round to 1 decimal point |
|  | smoker\_temp\_change = round(smoker\_deque\_item1\_temp - smoker\_current\_temp, 1) |
|  |  |
|  | # set up smoker alert |
|  | if smoker\_temp\_change >= smoker\_alert\_limit: |
|  | print(f">>> Smoker alert! The temperature of the smoker has decreased by 15 F or more in 2.5 min (or 5 readings). \n Smoker temp decrease = {smoker\_temp\_change} degrees F = {smoker\_deque\_item1\_temp} - {smoker\_current\_temp}") |
|  |  |
|  |  |
|  | def foodA\_callback(ch, method, properties, body): |
|  | """ Define behavior on getting a message in the 02-food-A queue. |
|  | Monitor food A temperature. Send an alert if the temp of food A changes (+/-) 1 F or less in 10 min (or 20 readings). """ |
|  | # receive & decode the binary message body to a string |
|  | print(f" [x] Received {body.decode()} on 02-food-A") |
|  | # simulate work |
|  | time.sleep(5) |
|  | # basic\_ack - acknowledge the message was received and processed (now it can be deleted from the queue) |
|  | ch.basic\_ack(delivery\_tag=method.delivery\_tag) |
|  |  |
|  | # create food A deque to store x amount of recent messages |
|  | # add new message to our food A deque |
|  | foodA\_deque.append(body.decode()) |
|  | # first item in deque (oldest) |
|  | foodA\_deque\_item1 = foodA\_deque[0] |
|  | # split the oldest message in the deque by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our date and timestamp from 20 messages ago (10 mins ago) |
|  | # the second list item [1] is the temp from 20 messages ago (10 mins ago) |
|  | foodA\_deque\_item1\_split = foodA\_deque\_item1.split(", ") |
|  | # change to float and remove last 2 characters from string, which is the ']' character of our message |
|  | foodA\_deque\_item1\_temp = float(foodA\_deque\_item1\_split[1][:-1]) |
|  |  |
|  | # food current temp/current message code |
|  | foodA\_current\_timetemp = body.decode() |
|  | # split the current message by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our current date and timestamp being read in this message |
|  | # the second list item [1] is our current temp being read in this message |
|  | foodA\_current\_timetemp\_split = foodA\_current\_timetemp.split(", ") |
|  | # change temp to float and remove last 1 characters from string, which is the ']' character of our message |
|  | foodA\_current\_temp = float(foodA\_current\_timetemp\_split[1][:-1]) |
|  |  |
|  | # calculate food temperature change and round to 1 decimal point |
|  | foodA\_temp\_change = round(foodA\_current\_temp - foodA\_deque\_item1\_temp, 1) |
|  |  |
|  | # set up food stall alert - any temp change (+/-) |
|  | if abs(foodA\_temp\_change) <= food\_stall\_alert\_limit: |
|  | print(f">>> Food stall alert! The temperature of food A has changed 1 F or less in 10 min (or 20 readings). \n Food A temp change = {foodA\_temp\_change} degrees F = {foodA\_current\_temp} - {foodA\_deque\_item1\_temp}") |
|  |  |
|  | def foodB\_callback(ch, method, properties, body): |
|  | """ Define behavior on getting a message in the 03-food-B queue. |
|  | Monitor food B temperature. Send an alert if the temp of food B changes (+/-) 1 F or less in 10 min (or 20 readings). """ |
|  | # receive & decode the binary message body to a string |
|  | print(f" [x] Received {body.decode()} on 03-food-B") |
|  | # simulate work |
|  | time.sleep(5) |
|  | # basic\_ack - acknowledge the message was received and processed (now it can be deleted from the queue) |
|  | ch.basic\_ack(delivery\_tag=method.delivery\_tag) |
|  |  |
|  | # create food B deque to store x amount of recent messages |
|  | # add new message to our food B deque |
|  | foodB\_deque.append(body.decode()) |
|  | # first item in deque (oldest) |
|  | foodB\_deque\_item1 = foodB\_deque[0] |
|  | # split the oldest message in the deque by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our date and timestamp from 20 messages ago (10 mins ago) |
|  | # the second list item [1] is the temp from 20 messages ago (10 mins ago) |
|  | foodB\_deque\_item1\_split = foodB\_deque\_item1.split(", ") |
|  | # change to float and remove last 2 characters from string, which is the ']' character of our message |
|  | foodB\_deque\_item1\_temp = float(foodB\_deque\_item1\_split[1][:-1]) |
|  |  |
|  | # food current temp/current message code |
|  | foodB\_current\_timetemp = body.decode() |
|  | # split the current message by the delimiter ", " and put data into list form |
|  | # the first list item [0] is our current date and timestamp being read in this message |
|  | # the second list item [1] is our current temp being read in this message |
|  | foodB\_current\_timetemp\_split = foodB\_current\_timetemp.split(", ") |
|  | # change temp to float and remove last 1 characters from string, which is the ']' character of our message |
|  | foodB\_current\_temp = float(foodB\_current\_timetemp\_split[1][:-1]) |
|  |  |
|  | # calculate food temperature change and round to 1 decimal point |
|  | foodB\_temp\_change = round(foodB\_current\_temp - foodB\_deque\_item1\_temp, 1) |
|  |  |
|  | # set up food stall alert - any temp change (+/-) |
|  | if abs(foodB\_temp\_change) <= food\_stall\_alert\_limit: |
|  | print(f">>> Food stall alert! The temperature of food B has changed 1 F or less in 10 min (or 20 readings). \n Food B temp change = {foodB\_temp\_change} degrees F = {foodB\_current\_temp} - {foodB\_deque\_item1\_temp}") |
|  |  |
|  | ######################################################## |
|  |  |
|  | # define a main function to run the program |
|  |  |
|  | def main(host: str, qn: str): |
|  | """ Continuously listen for task messages on a named queue.""" |
|  |  |
|  | # when a statement can go wrong, use a try-except block |
|  | try: |
|  | # try this code, if it works, keep going |
|  | # create a blocking connection to the RabbitMQ server |
|  | connection = pika.BlockingConnection(pika.ConnectionParameters(host)) |
|  |  |
|  | # except, if there's an error, do this |
|  | except Exception as e: |
|  | print() |
|  | print("ERROR: connection to RabbitMQ server failed.") |
|  | print(f"Verify the server is running on host={host}.") |
|  | print(f"The error says: {e}") |
|  | print() |
|  | sys.exit(1) |
|  |  |
|  | try: |
|  | # use the connection to create a communication channel |
|  | # need one channel per consumer |
|  | channel = connection.channel() |
|  |  |
|  | # use the channel to declare a durable queue (1 per queue) |
|  | # 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 |
|  | channel.queue\_declare(queue=smoker\_queue, durable=True) |
|  | channel.queue\_declare(queue=foodA\_queue, durable=True) |
|  | channel.queue\_declare(queue=foodB\_queue, durable=True) |
|  |  |
|  | # The QoS level controls the # of messages that can be in-flight (unacknowledged by the consumer) at any given time. |
|  | # Set the prefetch count to one to limit the number of messages being consumed and processed concurrently. |
|  | # This helps prevent a worker from becoming overwhelmed and improve the overall system performance. |
|  | # prefetch\_count = Per consumer limit of unaknowledged messages |
|  | channel.basic\_qos(prefetch\_count=1) |
|  |  |
|  | # configure the channel to listen on a specific queue, |
|  | # use the callback function named callback, |
|  | # we use the auto\_ack for this assignment |
|  | channel.basic\_consume(queue=smoker\_queue, on\_message\_callback=smoker\_callback, auto\_ack=False) |
|  | channel.basic\_consume(queue=foodA\_queue, on\_message\_callback=foodA\_callback, auto\_ack=False) |
|  | channel.basic\_consume(queue=foodB\_queue, on\_message\_callback=foodB\_callback, auto\_ack=False) |
|  |  |
|  | # print a message to the console for the user |
|  | print(" [\*] Ready for work. To exit press CTRL+C") |
|  |  |
|  | # start consuming messages via the communication channel |
|  | channel.start\_consuming() |
|  |  |
|  | # except, in the event of an error OR user stops the process, do this |
|  | except Exception as e: |
|  | print() |
|  | print("ERROR: something went wrong.") |
|  | print(f"The error says: {e}") |
|  | sys.exit(1) |
|  | except KeyboardInterrupt: |
|  | print() |
|  | print(" User interrupted continuous listening process.") |
|  | sys.exit(0) |
|  | finally: |
|  | print("\nClosing connection. Goodbye.\n") |
|  | connection.close() |
|  |  |
|  | ######################################################## |
|  |  |
|  | # Run program |
|  |  |
|  | # Standard Python idiom to indicate main program entry point |
|  | # This allows us to import this module and use its functions |
|  | # without executing the code below. |
|  | # If this is the program being run, then execute the code below |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | # call the main function with the information needed |
|  | main(host, smoker\_queue) |
|  | main(host, foodA\_queue) |
|  | main(host, foodB\_queue) |