

AWS Data Lake

This document goes through the AWS setup required to facilitate uploading of files and its subsequent insertion into a PostgreSQL database instance over AWS cloud services.

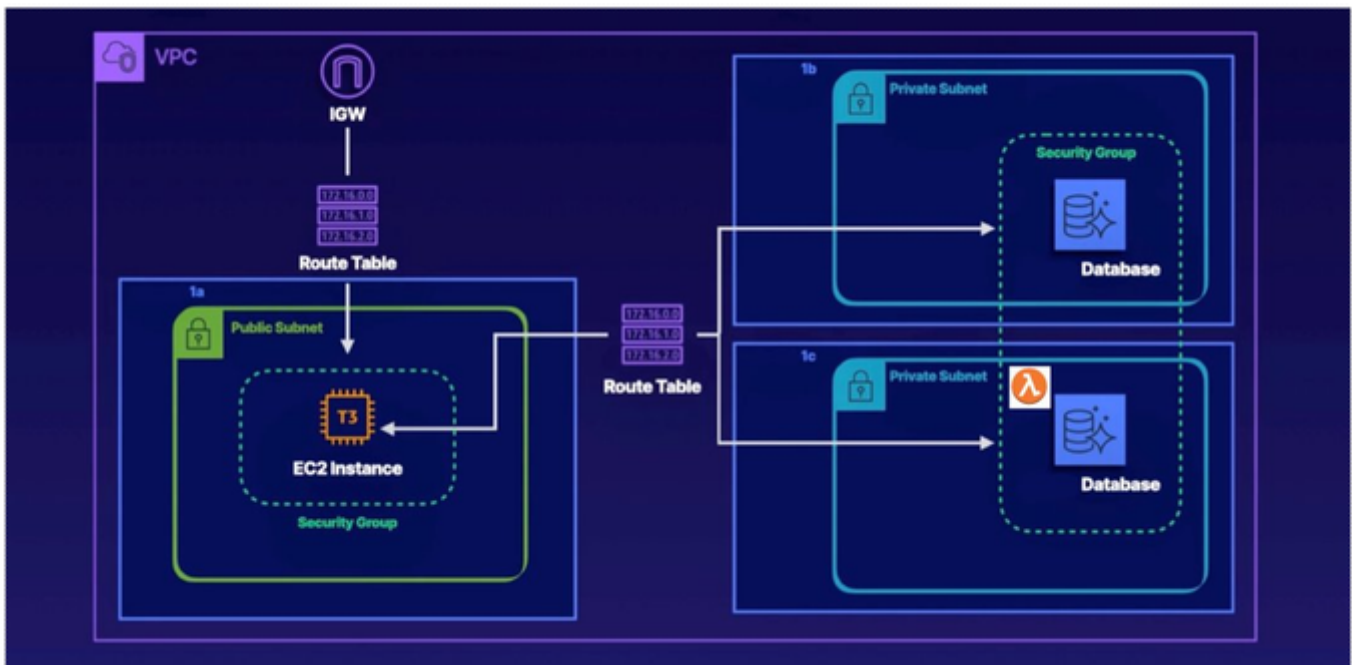
Login Page

[Amazon Web Services Sign-In](#)

Permissions

- Ensure that your AWS account has access to the following services:
 - S3
 - RDS
 - Lambda
 - SQS
 - IAM

Overall Architecture



We are going to setup an environment with the following components to host EC2 and databases for public access:

1. VPC (Virtual Private Cloud)
2. Subnets
 - a. One public
 - b. Two private
3. Internet Gateway
4. Route tables
5. Security Groups

Creating VPC and Subnets

- Navigate to **VPC** page in the AWS management console
- Click on **Create VPC**
- Input a name (eg. sparkdev-vpc-01) and under IPv4 CIDR input 10.0.0.0/16
- Navigate to **Subnets** sub-directory on the left-hand side of the **VPC** page
- Click on **Create subnet** and specify the following details:

- **VPC ID:** Select the VPC previously created (i.e. sparkdev-vpc-01)
- **Subnet name:** This will be used as the public subnet, give it a valid name eg. sparkdev-public-subnet-1
- **Availability zone:** Asia Pacific (Singapore) / ap-southeast-1a
- **IPv4 Block:** 10.0.1.0/24
- Instead of creating subnet, we select the **Add new subnet** option
 - Name should include "private-subnet-1"
 - **Availability zone:** Asia Pacific (Singapore) / ap-southeast-1b
 - **IPv4 Block:** 10.0.2.0/24
- Select **Add new subnet** to create the final subnet
 - Name should include "private-subnet-2"
 - **Availability zone:** Asia Pacific (Singapore) / ap-southeast-1c
 - **Pv4 Block:** 10.0.3.0/24
- Click on the **Create subnet** option at the bottom right

Configuring subnets' route tables

- Navigate to **Internet gateways** in the **VPC** page, click **Create gateway**, give it a name
- Once created, click **Options** and attach to the **VPC** previously created
- Navigate to the public subnet created previously (**VPC Subnets sparkdev-public-subnet-1**)
- Click **Actions Modify auto-assign IP Enable auto-assign IPv4** then save
- Click on **Route table** and select the route table to enter the route table page
- Under **Routes**, click on **Edit routes**.
- Add route with destination 0.0.0.0/0 and target the internet gateway just created
- Go back to public subnet page, and click on **Network ACL**, then click on network id
- Edit **Inbound rules** and **Outbound rules** by changing the first rule (the Allow rule) to **Type: All TCP**
- Go to **Route table** page and click **Create route table**
- Give it a name eg. sparkdev-private-route-table, and create
- Go to both of the private subnets previously created **Route table Edit route table associations**
- Under **Route table ID**, select the sparkdev-private-route-table previously created and **Save**

Subnets (1/7) [Info](#)

[Refresh](#) [Actions](#) [Create subnet](#)

	Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 addresses	Availabi
<input type="checkbox"/>	sparkdev-private-s...	subnet-02bea778aa762d59b	Available	vpc-0558304ed309c77c7 spa...	10.0.2.0/24	-	249	ap-south
<input type="checkbox"/>	-	subnet-6ec47708	Available	vpc-ed32fd8b	172.31.16.0/20	-	4091	ap-south
<input type="checkbox"/>	-	subnet-a8dbb4f1	Available	vpc-ed32fd8b	172.31.0.0/20	-	4091	ap-south
<input type="checkbox"/>	sparkdev-public-su...	subnet-0405e9a7ee2c102d8	Available	vpc-0558304ed309c77c7 spa...	10.0.1.0/24	-	249	ap-south
<input type="checkbox"/>	-	subnet-9ceb48d4	Available	vpc-ed32fd8b	172.31.32.0/20	-	4091	ap-south
<input type="checkbox"/>	sparkdev-private-s...	subnet-02a3ac54acd83c728	Available	vpc-0558304ed309c77c7 spa...	10.0.3.0/24	-	250	ap-south
<input checked="" type="checkbox"/>	nick-public-subnet	subnet-02f42a799143aa4f5	Available	vpc-0558304ed309c77c7 spa...	10.0.4.0/24	-	251	ap-south

subnets-02f42a799143aa4f5 / nick-public-subnet

[Details](#) [Flow logs](#) [Route table](#) [Network ACL](#) [CIDR reservations](#) [Sharing](#) [Tags](#)

Route table: **rtb-069f09ddd4880839b** [Edit route table association](#)

Routes (2)

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-0c6bbba4e9d4166b9

Note: The Internet gateway ([igw-0c6bbba4e9d4166b9](#)) is added in the table.

Configuring security groups

- Navigate to **Security Groups** page under **VPC > Security**

- Create a security group (eg name. Postgres-SG) with VPC specified as the one created above (i.e. sparkdev-vpc). Add an inbound rule with **Type:** PostgreSQL and **Source:** 10.0.0.0/16
- Create another security group (eg name. SSH-SG) with the same VPC as above and add an inbound rule with **Type:** SSH and **Source:** 0.0.0.0/0

Creating EC2 instance

- Navigate to **EC2** page in the AWS management console
- Select instances and **Launch instance** with the following specifications:
 - AMI - Amazon Linux 2 (Free Tier)
 - Instance type - Free Tier option then click on **Configure Instance Details**
 - Network - Select VPC previously created (i.e. sparkdev-vpc)
 - Subnet - Select the public subnet with internet gateway attached
 - Leave as default the next few pages, and under configure security groups, choose **Select an existing security group** and select the SSH-SG previously created
 - Select **Review and Launch > Launch**
 - When prompted with Key-Pair window, select **Create new key pair**, give it a name and click download key pair
 - **Save** your key pair file (required to access EC2 instance)
 - **Launch instance**

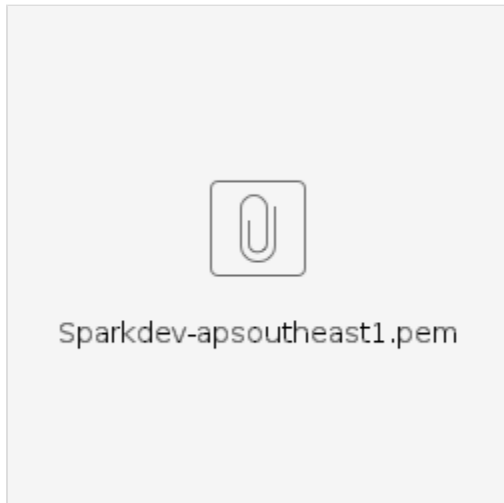
Creating RDS database instance

- Navigate to **RDS** page in the AWS management console
- First, on the left hand side of the page, click on **Subnet groups**
- Click on **Create DB subnet group** with the following specifications:
 - Name: Something appropriate (eg. PostgreSQL-RDS)
 - VPC: The VPC created previously (i.e. sparkdev-vpc)
 - Availability zones: The zones assigned to the private subnets (ap-southeast-1b and 1c)
 - Subnets: Both private subnets
- Now, navigate back to **Databases** in the **RDS** page
- Click on **Create Database**
- In the creation page, specify the following details:
 - **Choose a database creation method:** Standard create
 - **Engine options:** PostgreSQL
 - **Version:** Specify according to requirements, or just leave it as the default value
 - **Templates:** Free tier, unless otherwise needed
 - **Settings:** Specify details as needed. Store master username, password safely
 - **DB instance class:** Burstable classes (db.t2.micro), unless otherwise needed
 - **Storage:** default, unless otherwise needed
 - **Connectivity:**
 - VPC: The VPC created previously (i.e. sparkdev-vpc)
 - Subnet group: The group that was just created (i.e. PostgreSQL-RDS)
 - No public access
 - VPC Security group: De-select default and select the one previously created (i.e. Postgres-SG)
 - **Database authentication:** Password, unless otherwise needed.
 - **Additional configuration:** Include some database name, unless otherwise needed.

Connecting to RDS instance with DBeaver

- Open DBeaver Database New Database Connection
- Specify the following details for secure connection:
 - Host: Insert the rds database endpoint obtained from the RDS AWS management console page
 - Database: Some name, or the name specified when creating RDS instance (postgres etc.)
 - Username + Password: The master user and pass set during RDS creation (user: sparkdev, pass: Welcome2021!)
- Next, click the SSH header and specify the following:
 - Host/IP: Insert the public IPv4 address or the public IPv4 DNS
 - Username: Insert "ec2-user"
 - Authentication method: Public key and provide path to .pem file (default one below) created earlier for ec2 instance

- Click OK and the database should be available in the DBeaver platform



Creating SQS queue

- Navigate to **SQS** page in the AWS management console
- Click on **Create queue**
- In the creation page, add in a queue name (eg. s3_toLambda_queue) and leave the other specifications as default

Creating S3 bucket

- Navigate to **S3** page in the AWS management console
- Click on **Create bucket**
- Give the bucket a name, and choose the desired region, eg. Asia Pacific (Singapore)ap-southeast-1
- Click **Create/ Save**
- Select the newly created bucket, and navigate to **Objects > Create folder**
- Create 3 folders, **uploads**, **processed** and **errors** to store files that have been uploaded and subsequently processed, or failed to process by the lambda respectively

The screenshot displays the AWS Management Console for the 'sparkdev-file-bucket' in the 'ap-southeast-1' region. The left sidebar shows the navigation menu with 'Amazon S3' selected. The main content area shows the following settings:

- Server access logging:** Log requests for access to your bucket. [Learn more](#) [Edit]
- AWS CloudTrail data events:** Configure CloudTrail data events to log Amazon S3 object-level API operations in the CloudTrail console. [Learn more](#) [Configure in CloudTrail]
- Event notifications (2):** Send a notification when specific events occur in your bucket. [Learn more](#) [Edit] [Delete] [Create event notification]
- Transfer acceleration:** Use an accelerated endpoint for faster data transfers. [Learn more](#) [Edit]

The 'Event notifications' section contains a table with the following data:

<input type="checkbox"/>	Name	Event types	Filters	Destination type	Destination
<input type="checkbox"/>	FileUploadTriggerSqsEvent	All object create events	uploads, csv	SQS queue	s3_toLambda_queue ↗
<input type="checkbox"/>	eodEventPut	All object create events	eod_dir, csv	SQS queue	s3_toLambda_queue ↗

A red error box is visible under the 'AWS CloudTrail data events' section, stating: 'You don't have permission to get AWS CloudTrail data events details. You or your AWS administrator must update your IAM permissions to allow cloudtrail:DescribeTrails. After you obtain the necessary permission, choose Refresh. [Learn more about Identity and access management in Amazon S3](#) [↗](#)'. There is a 'Refresh' button next to the message.

Note: Event notification to queue (SQS)

Configuring SQS permissions and access policy

- Navigate to **SQS** page in the AWS management console
- Click on the previously created queue, and navigate to the **Access policy** tab
- Add in the necessary permissions for S3-SQS access permissions or use the following template:

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<INSERT ACCOUNT ID>:root"
      },
      "Action": "SQS:*",
      "Resource": "<INSERT SQS QUEUE ARN>"
    },
    {
      "Sid": "__sender_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<INSERT ACCOUNT ID>:root",
          "arn:aws:iam::<INSERT ACCOUNT ID>:user/<INSERT IAM USERNAME>"
        ]
      },
      "Action": "SQS:SendMessage",
      "Resource": "<INSERT SQS QUEUE ARN>"
    },
    {
      "Sid": "__receiver_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<INSERT ACCOUNT ID>:root",
          "arn:aws:iam::<INSERT ACCOUNT ID>:user/<INSERT IAM USERNAME>"
        ]
      },
      "Action": [
        "SQS:ChangeMessageVisibility",
        "SQS:DeleteMessage",
        "SQS:ReceiveMessage"
      ],
      "Resource": "<INSERT SQS QUEUE ARN>"
    },
    {

```

```

    "Sid": "example-statement-ID",
    "Effect": "Allow",
    "Principal": {
        "Service": "s3.amazonaws.com"
    },
    "Action": "SQS:SendMessage",
    "Resource": "<INSERT SQS QUEUE ARN>",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "<INSERT S3 BUCKET ARN>"
        }
    }
}
]
}

```

Configuring S3 Event Notification to SQS

- Navigate back to **S3** page in the AWS management console, click on the bucket name
- In the upper half of the page, change to the **Properties** tab
- Scroll down and under **Event notifications**, select **Create event notification**
- In the creation page, specify the following details:
 - Give the event some meaningful name eg. "FileUploadTriggerSqsEvent"
 - **Prefix:** uploads
 - **Suffix:** .csv
 - **Event types:** tick the "All object create events" box
 - **Destination:** Click SQS queue, then from the dropdown, select your previously created queue

Creating IAM Execution Role for Lambda

- Navigate to **IAM** page in the AWS management console
- Click on **Roles > Create Role**
- Click on **Lambda > Next: Permissions**
- Add AWSLambdaVPCLambdaAccessExecutionRole, AWSLambdaSQSQueueExecutionRole and AmazonS3FullAccess
- Click **Next > Next** and give the role a name eg. lambda-vpc-execution-role and **Create role**

Creating Lambda function

Specified below are two different approaches to configure the AWS Lambda function. Choose either.

Python

- On your local environment, create a folder and give it a name eg. py-postgres
- In the folder, create a file db_util.py
- Open the folder and paste the following code, substituting the "<>" with the appropriate parameters:

```

import psycopg2
import csv
import logging
import datetime
import json
import boto3

```

```

import time
from functools import wraps

db_host = "<INSERT RDS ENDPOINT HERE>"
db_port = 5432
db_name = "<INSERT DB NAME>"
db_user = "<INSERT MASTER USERNAME>"
db_pass = "<INSERT MASTER PASSWORD>"
db_table = "fx.fx_order" # or any other table name
column_names = "<COLUMN_1, COLUMN_2, .... etc>" # in order of table
columns

# sets up postgres connection with RDS db instance using psycopg2
module
def make_conn():

    conn = None

    try:

        conn = psycopg2.connect("dbname='%s' user='%s' host='%s'
password='%s'" % (db_name, db_user, db_host, db_pass))

    except Exception as e:

        print("I am unable to connect to the database" + str(e))

    return conn

# executes query and fetches server response to query
def fetch_data(conn, query):

    result = []
    print("Now executing: %s" % (query))
    cursor = conn.cursor()
    cursor.execute(query)

    # subclass JSONEncoder - changes to datetime to str during fetch
    class DateTimeEncoder(json.JSONEncoder):
        #Override the default method
        def default(self, obj):
            if isinstance(obj, (datetime.date, datetime.datetime)):
                return obj.isoformat()

    raw = cursor.fetchall()
    cursor.close()

    for line in raw:
        line = DateTimeEncoder().encode(line)

```

```

        result.append(line)

    return result

# creates a profile for a method to display its execution time
def profile(fn):

    @wraps(fn)
    def inner(*args, **kwargs):
        fn_kwargs_str = ', '.join(f'{k}={v}' for k, v in kwargs.items())
        print(f'\n{fn.__name__}({fn_kwargs_str})')

        # Measure time
        t = time.perf_counter()
        retval = fn(*args, **kwargs)
        elapsed = time.perf_counter() - t
        print(f'Time    {elapsed:0.4}')

        return retval

    return inner

# uses csv_obj from s3, pushing data to RDS db table
@profile
def csv_to_table(csv_obj, conn):

    body = csv_obj['Body']

    flag = False

    try:

        cursor = conn.cursor()

        # converts data to readable format
        data = (body.read().decode('utf-8')).splitlines()
        lines = csv.reader(data)

        # removes header column
        next(lines)

        for line in lines:

            line = tuple(line)
            st = str(line)

            # replacing all '' occurrences in csv file (empty cells) to
            NULL for sql queries to run without formatting issues
            st = st.replace("'", "NULL")

```



```

        # query inserts row into db table if primary key (id) does
        not already exist, and updates the row value otherwise
        cursor.execute("INSERT INTO {}{} VALUES {} ON CONFLICT (id)
        DO UPDATE SET {} = {};".format(db_table, column_names, st,
        column_names, st))

        # committing sql query execution changes
        conn.commit()

        cursor.close()
        logger.info("Loaded data into {}".format(db_table))

        flag = True

    except Exception as e:

        logger.info("csv_to_table Error: {}".format(str(e)))

    return flag

```

- In the same folder, create another file `postgres_test.py` and paste the following code:

```

import sys
import logging
import psycopg2

import boto3
import requests
import os
import csv
import json
import ast

s3_client = boto3.client('s3')
s3_resource = boto3.resource('s3')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

from db_util import make_conn, fetch_data, csv_to_table, db_table

# handles events passed from SQS trigger
def lambda_handler(event, context):

    return 1
    # connecting to the RDS db instance
    conn = make_conn()

    if event:

```

```

try:

    print(str(event))

    # transforming event parameter and extracting s3 bucket
name and key path
    record = event['Records'][0]
    body = record['body']

    # using ast module to transform 'body' in event from str to
dict
    body_contents_in_dict_format = ast.literal_eval(body)
['Records'][0]

    bucket = body_contents_in_dict_format['s3']['bucket']
['name']
    key_path = body_contents_in_dict_format['s3']['object']
['key']

    # using boto3 s3 client to extract s3.Object
    csv_obj = s3_client.get_object(Bucket=bucket, Key=key_path)

    # pusing data to rds
    is_pushed_successfully = csv_to_table(csv_obj, conn)

    # retrieving csv file name
    object_name = key_path[8:]

    if pushed_successfully:

        # moving csv_file from /uploads folder to /processed
folder in S3 once processed
        s3_resource.Object(bucket, 'processed/{}'.format
(object_name)).copy_from(CopySource='/{}'.format(bucket,
object_name))
        s3_resource.Object(bucket, 'uploads/{}'.format
(object_name)).delete()

    else:

        # moving csv_file to /errors if data not pushed
successfully
        s3_resource.Object(bucket, 'errors/{}'.format
(object_name)).copy_from(CopySource='/{}'.format(bucket,
object_name))
        s3_resource.Object(bucket, 'uploads/{}'.format
(object_name)).delete()

    #s3_client.copy_object(Bucket=bucket, CopySource=key_path,
Key="/processed/{}".format(bucket, object_name))

```

```

        #s3_client.delete_object(Bucket=bucket, Key=key_path)

    except Exception as e:

        logger.info("lambda_handler has exception: {}".format(str
(e)))

    else:

        logger.info('No event')

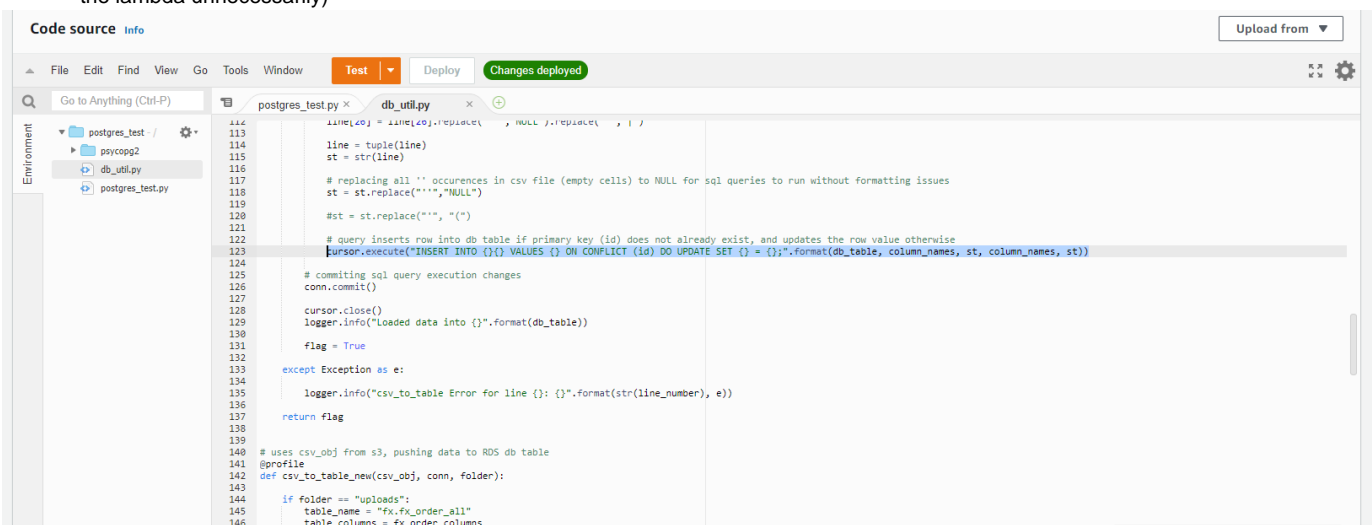
    query_cmd = 'select count(*) from fx.fx_order'
    result = fetch_data(conn, query_cmd)

    conn.close()

    return result

```

- Next, we need to add in the psycopg2 module to connect to the postgres database since AWS Lambda does not provide local access to some modules
- Go to <https://github.com/jkehler/awslambda-psycopg2> and download the appropriate psycopg2 folder depending on the python version you plan to use. For reference, we use the psycopg2-3.7 for python 3.7 in this example
- Download the psycopg2-3.7 (or otherwise) and rename it to psycopg2 before placing it into the local folder (i.e. py-postgres) created earlier
- ZIP the folder
- Return to AWS management console and navigate to the **Lambda** page
- Click **Create function > Author from scratch**
- Choose a python runtime (eg. Python 3.7 in this case)
- Expand **Permissions** tab, select **Use an existing role** and select the previously created role i.e. lambda-vpc-execution-role
- Under **Advanced settings**, select the VPC previously created i.e. sparkdev-vpc-01
- Specify the 2 private subnets, and the PostgreSQL security group created earlier
- Click **Create function**
- Wait for function to be successfully created, then under the **Code** section from within the function page, click on the **Upload from** button on the right hand side, select .zip and choose the zipped folder (i.e. py-postgres.zip) created earlier
- Change the handler to <INSERT .py file name>.<INSERT function name> eg. (postgres_test.lambda_handler)
- Next go to the lambda function page created > configuration > change visibility timeout to 5 minutes (or otherwise). This is due to the fact that AWS Lambda will auto re-process the event given its processing is not complete within timeout duration. Thus, for bigger CSVs (>20mb), a lower timeout time may cause repeated processing (which wont affect db but will invoke additional memory costs for running the lambda unnecessarily)



```

Code source info
Upload from

File Edit Find View Go Tools Window Test Deploy Changes deployed

Go to Anything (Ctrl-P)
Environment
  postgres_test - /
  psycopg2
  db_util.py
  postgres_test.py

114 line_noj = line_noj.replace(' ', NULL ).replace(' ', )
115
116 line = tuple(line)
117 st = str(line)
118
119 # replacing all '' occurrences in csv file (empty cells) to NULL for sql queries to run without formatting issues
120 st = st.replace('','NULL')
121
122 #st = st.replace(' ','(')
123
124 # query inserts row into db table if primary key (id) does not already exist, and updates the row value otherwise
125 cursor.execute("INSERT INTO {}() VALUES {} ON CONFLICT (id) DO UPDATE SET {} = {};".format(db_table, column_names, st, column_names, st))
126
127 # committing sql query execution changes
128 conn.commit()
129
130 cursor.close()
131 logger.info("Loaded data into {}".format(db_table))
132
133 flag = True
134
135 except Exception as e:
136     logger.info("csv_to_table Error for line {}: {}".format(str(line_number), e))
137
138 return flag
139
140 # uses csv_obj from s3, pushing data to RDS db table
141 @profile
142 def csv_to_table_new(csv_obj, conn, folder):
143
144     if folder == "uploads":
145         table_name = "fx.fx_order_all"
146         table_columns = "fx_order_columns"

```

```
147         on_conflict_string = "(id)"
148     elif folder == "cod_dir":
149         table_name = "ev_and_conf_data"
```

147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

Note: We use the ON CONFLICT to either insert or update

Configure VPC Endpoint for Lambda to access S3

- Navigate to **VPC > Endpoints > Create endpoint** in the AWS management console
- Tick AWS services, search for s3 and click on the gateway option
- Select the sparkdev-vpc in the vpc dropdown and select the route table that is associated with the 2 private subnets (where the lambda is situated)
- Grant full access unless otherwise needed and click **Create endpoint**