```
#!/usr/bin/python3.6
from botocore.vendored import requests
import urllib3
import boto3
from botocore.exceptions import ClientError
import datetime
import json
import logging
import urllib.parse
logger = logging.getLogger()
logger.setLevel(logging.INFO)
iam_client = boto3.client('iam')
requests = urllib3.PoolManager()

def list_access_key(user, days_filter, status_filter):
    keydetails=iam_client.list_access_keys(UserName=user)
    key_details={}
    user_iam_details=[]

    # Some user may have 2 access keys.
    for keys in keydetails['AccessKeyMetadata']:
        if (days:=time_diff(keys['CreateDate'])) >= days_filter and keys['Status']==status_filter:
            key_details['UserName'] = keys['UserName']
            key_details['AccessKeyId'] = keys['AccessKeyId']
            key_details['days'] = days
            key_details['status'] = keys['Status']
            user_iam_details.append(key_details)
            key_details={}

    return user_iam_details

def get_user_info(username):
    print("username " + username)
    userdetails=iam_client.get_user(UserName=username)
    tags= userdetails['User']['Tags']
    for tag in tags:
        if 'email' in tag['Key']:
            tagvalue = tag['Value']
    logger.info(f"Email Address {tagvalue}")

    return tagvalue

def time_diff(keycreatedtime):
    now=datetime.datetime.now(datetime.timezone.utc)
    diff=now-keycreatedtime
    return diff.days


def lambda_handler(event, context):
     ssm_client = boto3.client('ssm')
     ssm_response = ssm_client.get_parameters(
       Names=[
            'Iam_user_list_for_key_rotation',
          ])
```

```
    users_list_not_rotate = ssm_response['Parameters'][0]['Value']

    details = iam_client.list_users(MaxItems=100)
    print(details)
    users = details['Users']

    for user in users:
     user_name=user['UserName']

     if user_name not in users_list_not_rotate:

      user_iam_details=list_access_key(user=user_name,days_filter=90,status_filter='Active')
      for _ in user_iam_details:
       emailid = get_user_info(username=_['UserName'])
       send_key_rotation_reminder(emailid, user_name)
      else:
       print("can not rotate access key for user " + user_name)

def send_key_rotation_reminder(emailid, user_name):

  secret_client = boto3.client("secretsmanager")
  secret_name = "slack_bot_token"
  try:
    get_secret_value_response = secret_client.get_secret_value(
            SecretId = secret_name
    )
  except ClientError as e:
    raise e
  else:
    if 'SecretString' in get_secret_value_response:
       secret = get_secret_value_response['SecretString']
    secret = json.loads(secret)
    secret_token = secret['slack_bot_token']

  slack_baseUrl = "https://slack.com/api/"
  email_to_lookup = emailid
  slack_lookup_url = slack_baseUrl + "users.lookupByEmail?email=" + email_to_lookup
  slack_bot_token = "Bearer " + secret_token

  payload = ""
  lookup_headers = {"Authorization": slack_bot_token}

 # Retrieve Slack User Id from Email
  response = requests.request(
      "GET", slack_lookup_url, headers=lookup_headers
  )
  print(response.data)

  slack_lookup_response = json.loads(response.data)
  slack_user_id = slack_lookup_response["user"]["id"]
  slack_display_name = slack_lookup_response["user"]["profile"]["display_name"]
  print("Slack User Id: " + slack_user_id)
  print("Slack Display Name: " + slack_display_name)
 # Send Slack Message
```

```
   slack_message_text = "Hello "+slack_display_name+"! Your AWS Credentials have been
expired your keys will be get deleted tomorrow. So, Please create new keys using this link:
https://console.aws.amazon.com/iam/home#/users/" +user_name+"?
section=security_credentials"
   slack_message_encoded = urllib.parse.quote_plus(slack_message_text)
   slack_dm_url = (
    slack_baseUrl
    + "chat.postMessage?channel="
    + slack_user_id
    + "&text=" + slack_message_encoded + "&as_user=true"
   )
   payload = {"Email" : emailid}

   #encode_slack_dm_url= slack_dm_url.encode('utf-8')
   dm_headers = {"Accept": "application/x-www-form-urlencoded", "Authorization":
slack_bot_token}

   msg = json.dumps(payload)

   slack_dm_response = requests.request(
       "POST", slack_dm_url, headers=dm_headers
    )

   print(slack_dm_response.data)

   return {
       'statusCode': 200,
       #'body': list_access_key(user=user,days_filter=0,status_filter='Active')
    }



############################################################

Delete Key:

#!/usr/bin/python3.6
from botocore.vendored import requests
import urllib3
import boto3
from botocore.exceptions import ClientError
import datetime
import json
import logging
import urllib.parse
logger = logging.getLogger()
logger.setLevel(logging.INFO)
iam_client = boto3.client('iam')
requests = urllib3.PoolManager()

def list_access_key(user, days_filter, status_filter):
    keydetails = iam_client.list_access_keys(UserName=user)
    key_details = {}
    user_iam_details = []
```

```
    # Some user may have 2 access keys.
    for keys in keydetails['AccessKeyMetadata']:
        if (days:=time_diff(keys['CreateDate'])) >= days_filter and keys['Status'] == status_filter:
            key_details['UserName'] = keys['UserName']
            key_details['AccessKeyId'] = keys['AccessKeyId']
            key_details['days'] = days
            key_details['status'] = keys['Status']
            user_iam_details.append(key_details)
            key_details = {}

    return user_iam_details

def get_user_info(username):
    print("username " + username)
    userdetails = iam_client.get_user(UserName=username)
    tags = userdetails['User']['Tags']
    for tag in tags:
        if 'email' in tag['Key']:
            tagvalue = tag['Value']
    logger.info(f"Email Address {tagvalue}")

    return tagvalue

def time_diff(keycreatedtime):
    now = datetime.datetime.now(datetime.timezone.utc)
    diff = now-keycreatedtime
    return diff.days

def disable_key(access_key, username):
    try:
        iam_client.update_access_key(UserName=username, AccessKeyId=access_key,
Status="Inactive")
        print(access_key + " has been disabled.")
    except ClientError as e:
        print("The access key with id %s cannot be found" % access_key)
def delete_key(access_key, username):
    try:
        iam_client.delete_access_key(UserName=username, AccessKeyId=access_key)
        print (access_key + " has been deleted.")
    except ClientError as e:
        print("The access key with id %s cannot be found" % access_key)
def lambda_handler(event, context):
    ssm_client = boto3.client('ssm')
    ssm_response = ssm_client.get_parameters(
      Names=[
            'Iam_user_list_for_key_rotation',
        ])
    users_list_not_rotate = ssm_response['Parameters'][0]['Value']
    details = iam_client.list_users(MaxItems=100)
    print(details)
    users = details['Users']

    for user in users:
```

```
    user_name = user['UserName']

    if user_name not in users_list_not_rotate:

     user_iam_details = list_access_key(user=user_name,days_filter=91,status_filter='Active')
     for _ in user_iam_details:
       emailid = get_user_info(username=_['UserName'])
       disable_key(access_key=_['AccessKeyId'], username=_['UserName'])
       delete_key(access_key=_['AccessKeyId'], username=_['UserName'])
       send_key_deletion_reminder(emailid)
    else:
     print("can not rotate access key for user " + user_name)

def send_key_deletion_reminder(emailid):
  secret_client = boto3.client("secretsmanager")
  secret_name = "slack_bot_token"
  try:
    get_secret_value_response = secret_client.get_secret_value(
            SecretId = secret_name
    )
  except ClientError as e:
    raise e
  else:
    if 'SecretString' in get_secret_value_response:
      secret = get_secret_value_response['SecretString']
    secret = json.loads(secret)
    secret_token = secret['slack_bot_token']

  slack_baseUrl = "https://slack.com/api/"
  email_to_lookup = emailid
  slack_lookup_url = slack_baseUrl + "users.lookupByEmail?email=" + email_to_lookup
  slack_bot_token = "Bearer " + secret_token

  payload = ""
  lookup_headers = {"Authorization": slack_bot_token}

 # Retrieve Slack User Id from Email
  response = requests.request(
      "GET", slack_lookup_url, headers=lookup_headers
  )
  print(response.data)

  slack_lookup_response = json.loads(response.data)
  slack_user_id = slack_lookup_response["user"]["id"]
  slack_display_name = slack_lookup_response["user"]["profile"]["display_name"]
  print("Slack User Id: " + slack_user_id)
  print("Slack Display Name: " + slack_display_name)
 # Send Slack Message
  slack_message_text = "Hello "+slack_display_name+"! Your old AWS Credentials have been
deleted."
  slack_message_encode = urllib.parse.quote_plus(slack_message_text)
  slack_dm_url = (
   slack_baseUrl
   + "chat.postMessage?channel="
```

```
    + slack_user_id
    + "&text=" + slack_message_encode + "&as_user=true"
   )
   payload = {"Email" : emailid}

   dm_headers = {"Accept": "application/x-www-form-urlencoded", "Authorization":
slack_bot_token}

   msg = json.dumps(payload)

   slack_dm_response = requests.request(
       "POST", slack_dm_url, headers=dm_headers
    )

   print(slack_dm_response.data)

   return {
       'statusCode': 200,
    }
```