Technical specifications - SpaceOdyssey automated radius updates

Setting up an e-mail service to receive the client's CSV

The first step is to find a way to receive emails with the CSV file securely where we have control of what comes in - for this, we could use AWS Simple Email Service or SES. This however needs domain ownership proof through DNS settings so we will discard that as it could create issues and delays.
The second option considered would be to use MailGun. Unfortunately this service is not free, so I've gone for using a Google script. The script would run once or twice during the weekdays (twice in case the client sends the csv late) and third time on Saturday 12am to update weekend radius.

1. Setting up Google Apps Script to monitor Gmail account

1.1. Create a new Gmail account for the purpose of receiving the weekly emails with CSV attached.

1.2. Go to Google Apps Script website https://script.google.com/home and create a new project named e.g. '*GmailCheckerSpaceOdyssey'.*

1.3. Define the function which will check Gmail account for emails with specific subject e.g. '*SpaceOdyssey - automated radius updates CSV'. Note that this query can be adjusted* - see the Google Apps Script written in JavaScript.

1.3.1. NB. It is important that the email subject is without any mistakes

1.4. Set up trigger to run the Google Apps Script by accessing GmailCheckerSpaceOdyssey project > Triggers > Create a new trigger > Choose which function to run in our case 'checkEmails' function > Select event source to be 'Time-driven' > Time based trigger will be 'Week timer' > Day of week 'Every Tuesday' >Time of day '5pm to 6pm'. Note that this is an example of when to run the GA Script and should be agreed on with the client.

1.4.1. We can set up second trigger for the following day (Wednesday) morning to make sure that the script will run even if the email will be sent late.

1.4.2. Third trigger would be timed to Saturday 12am to 1am, so the ad campaigns are updated with the weekend radius.

1.5. First function run will have to be authorised by running it manually and following prompt that will appear to authorise access to the Gmail account.

1.6. Test can be done by sending an email with correct subject and csv file and running the scrip manually, then checking execution logs to see if script run correctly and Google and Facebook ad accounts.

Google Apps Script - JavaScript

Processes the CSV attachment in the email and sends it as a payload to our Lambda function

```
function checkEmails() {
    var threads = GmailApp.search('subject:SpaceOdyssey - automated radius
updates CSV');

    if (threads.length > 0) {
        var thread = threads[threads.length - 1]; // Process the latest
thread that matches the query
        var messages = thread.getMessages();
        var message = messages[messages.length - 1]; // Process the latest
message in the thread

        var attachments = message.getAttachments();
        for (var i = 0; i < attachments.length; i++) {
            var attachment = attachments[i];
            if (attachment.getContentType() === 'text/csv') {
                var csvData = attachment.getDataAsString();

                var apiUrl =
'https://aws-api-gateway-endpoint-for-lambda-function';
                var payload = { "csvData": csvData };

                var options = {
                    'method': 'post',
                    'contentType': 'application/json',
                    'payload': JSON.stringify(payload)
                };

                UrlFetchApp.fetch(apiUrl, options);
            }
        }
    }
}
```

Google Apps Script function ‘checkEmail’ - documentation links

GmailApp class documentation - link

- GmailApp.search() - docs link
- thread.getMessages() - docs link
- message.isUnread() - docs link
- message.getAttachements() - docs link

- attachment.getConentType() - docs link
- attachment.getDataAsString() - docs link

UrlFetchApp class documentation -

- UrlFetchApp.fetch() - docs link

2. Setting up AWS API Gateway, S3 and Lambda function

In the next steps, we will go through set-up of AWS services to perform processing of the CSV data as well as storing sales data in an S3 bucket for later visualisation. We will assume that basic security concepts when using these services are satisfied, such as least-privilage IAM policy or keeping the S3 bucket private and not publicly accessible. When needed, these will be specified.

NB. Make sure all services are in the same region.

2.1. Create a new S3 bucket in AWS console named e.g. '*sales-data-spaceodyssey-bucket'.*

2.1.1. Set Bucket permissions and ensure the bucked is private

2.2. Create Lambda named e.g. *'ProcessCSV-spaceodyssey'.*

2.2.1. The runtime will be Python. *Note: choose the same version as programmed in e.g. 3.11.*

2.2.2. Create a new role with basic lambda permissions, then create a function.

2.2.3. Go to IAM roles, select newly created role > create a policy which satisfies the least-privilege approach .

2.2.4. Attach the created policy in IAM > Roles by selecting the Lambda role

2.2.5. Go to '*ProcessCSV-spaceodyssey'* lambda function and write the function code (delete the default code). The lambda function is written in Python. See the Lambda script Lambda - Python script.

2.3. Create and deploy API Gateway.

2.3.1. Click Create API and choose REST API - name e.g. 'CSVProcessingAPISpaceodyssey'.

2.3.2. Go to 'Resources' and create a new resource named e.g. *'csv-spaceodyssey'.*

2.3.3. Select the newly created resource and click 'Create method'. Method type will be POST, Integration type Lambda, Choose region - note it has to be the same as the Lambda function and finally choose our lambda '*ProcessCSV-spaceodyssey'* .

2.3.4. Click on Deploy API then choose *new stage* as a name e.g. 'prod' and click Deploy. Copy the URL endpoint which will be provided by API Gateway e.g. *'https://your-api-id.execute-api.region.amazonaws.com/prod/csv-spaceodyssey'.*

Google Ads:

We will assume that the client's ads account integration has already been dealt with and we have all the tokens necessary to make requests to their Google account API by the media activation team.

Lambda - Python script

```
import requests
import json
import boto3
from datetime import datetime

s3_client = boto3.client('s3')
S3_BUCKET_NAME = 'sales-data-spaceodyssey-bucket'

def lambda_handler(event, context):
    csv_data = event['csvData']
    csv_lines = csv_data.splitlines()
    raw_headers = csv_lines[0].split(',')
    headers = [header.strip().lower().replace(' ', '_').replace('(',
'').replace(')', '') for header in raw_headers]
    csv_list_of_dicts = []
    for line in csv_lines[1:]:
        fields = line.split(',')
        row_dict = {headers[i]: fields[i] for i in range(len(headers))}
        csv_list_of_dicts.append(row_dict)
    for row in csv_list_of_dicts:
        location = row['coordinates']
        weekday_radius = row['weekday_radius_m']
        weekend_radius = row['weekend_radius_m']
        action = row['action']
        sales = row['sales']
        latitude, longitude = parse_coordinates(location)
        if action.lower() == "boost":
            boost_ads(location, latitude, longitude, weekday_radius,
weekend_radius, sales)
        elif action.lower() == "exclude":
            exclude_ads(location, latitude, longitude)
        elif action.lower() == "regular":
            regular_ads(location, latitude, longitude, weekday_radius,
weekend_radius, sales)
    return {
        'statusCode': 200,
        'body': json.dumps('Ads distributed successfully')
    }
```

```
def parse_coordinates(location):
    latitude, longitude = map(float, location.split(','))
    return latitude, longitude

def boost_ads(location, latitude, longitude, weekday_radius, weekend_radius,
sales):
    google_access_token = "GOOGLE_ACCESS_TOKEN"
    facebook_access_token = "FACEBOOK_ACCESS_TOKEN"
    google_campaign_id = "GOOGLE_CAMPAIGN_ID"
    facebook_campaign_id = "FACEBOOK_CAMPAIGN_ID"
    google_ads_api_url =
f"https://googleads.googleapis.com/v6/customers/CUSTOMER_ID/campaigns:mutate
"
    facebook_ads_api_url =
f"https://graph.facebook.com/v11.0/{facebook_campaign_id}"
    google_headers = {
        'Authorization': f'Bearer {google_access_token}',
        'Content-Type': 'application/json'
    }
    facebook_headers = {
        'Authorization': f'Bearer {facebook_access_token}',
        'Content-Type': 'application/json'
    }
    current_day = datetime.now().weekday()
    radius = weekday_radius if current_day < 5 else weekend_radius
    google_payload = {
        "operations": [
            {
                "update": {
                    "resourceName":
f"customers/CUSTOMER_ID/campaigns/{google_campaign_id}",
                    "status": "ENABLED"
                },
                "updateMask": "status"
            },
            {
                "update": {
                    "resourceName":
f"customers/CUSTOMER_ID/campaignCriteria/{google_campaign_id}",
                    "proximity": {
                        "geoPoint": {
                            "latitudeInMicroDegrees": int(latitude * 1e6),
                            "longitudeInMicroDegrees": int(longitude * 1e6)
                        },
                        "radius": radius,
                        "radiusUnits": "METERS"
                    }
                },
```

```
                "updateMask": "proximity.radius,proximity.geoPoint"
            }
        ]
    }
    facebook_payload = {
        "name": f"Boosted Campaign for {location}",
        "targeting": {
            "geo_locations": {
                "custom_locations": [
                    {
                        "latitude": latitude,
                        "longitude": longitude,
                        "radius": (radius) / 1000,
                        "distance_unit": "kilometer"
                    }
                ]
            }
        },
        "status": "ACTIVE"
    }
    google_response = send_post_request(google_ads_api_url, google_headers,
google_payload)
    print(f"Google Ads response: {google_response}")
    facebook_response = send_post_request(facebook_ads_api_url,
facebook_headers, facebook_payload)
    print(f"Facebook Ads response: {facebook_response}")
    store_sales_data(location, sales, "boost")

def exclude_ads(location, latitude, longitude):
    google_access_token = "GOOGLE_ACCESS_TOKEN"
    facebook_access_token = "FACEBOOK_ACCESS_TOKEN"
    google_campaign_id = "GOOGLE_CAMPAIGN_ID"
    facebook_campaign_id = "FACEBOOK_CAMPAIGN_ID"
    google_ads_api_url =
f"https://googleads.googleapis.com/v6/customers/CUSTOMER_ID/campaigns:mutate
"
    facebook_ads_api_url =
f"https://graph.facebook.com/v11.0/{facebook_campaign_id}"
    google_headers = {
        'Authorization': f'Bearer {google_access_token}',
        'Content-Type': 'application/json'
    }
    facebook_headers = {
        'Authorization': f'Bearer {facebook_access_token}',
        'Content-Type': 'application/json'
    }
    google_payload = {
        "operations": [
```

```
            {
                "update": {
                    "resourceName":
f"customers/CUSTOMER_ID/campaigns/{google_campaign_id}",
                    "status": "PAUSED"
                },
                "updateMask": "status"
            }
        ]
    }
    facebook_payload = {
        "targeting": {
            "geo_locations": {
                "excluded_custom_locations": [
                    {
                        "latitude": latitude,
                        "longitude": longitude
                    }
                ]
            }
        },
        "status": "PAUSED"
    }
    google_response = send_post_request(google_ads_api_url, google_headers,
google_payload)
    print(f"Google Ads response: {google_response}")
    facebook_response = send_post_request(facebook_ads_api_url,
facebook_headers, facebook_payload)
    print(f"Facebook Ads response: {facebook_response}")
    store_sales_data(location, sales, "exclude")

def regular_ads(location, latitude, longitude, weekday_radius,
weekend_radius, sales):
    google_access_token = "GOOGLE_ACCESS_TOKEN"
    facebook_access_token = "FACEBOOK_ACCESS_TOKEN"
    google_campaign_id = "GOOGLE_CAMPAIGN_ID"
    facebook_campaign_id = "FACEBOOK_CAMPAIGN_ID"
    google_ads_api_url =
f"https://googleads.googleapis.com/v6/customers/CUSTOMER_ID/campaigns:mutate
"
    facebook_ads_api_url =
f"https://graph.facebook.com/v11.0/{facebook_campaign_id}"
    google_headers = {
        'Authorization': f'Bearer {google_access_token}',
        'Content-Type': 'application/json'
    }
    facebook_headers = {
        'Authorization': f'Bearer {facebook_access_token}',
```

```
        'Content-Type': 'application/json'
    }
    current_day = datetime.now().weekday()
    radius = weekday_radius if current_day < 5 else weekend_radius
    google_payload = {
        "operations": [
            {
                "update": {
                    "resourceName":
f"customers/CUSTOMER_ID/campaigns/{google_campaign_id}",
                    "status": "ENABLED"
                },
                "updateMask": "status"
            },
            {
                "update": {
                    "resourceName":
f"customers/CUSTOMER_ID/campaignCriteria/{google_campaign_id}",
                    "proximity": {
                        "geoPoint": {
                            "latitudeInMicroDegrees": int(latitude * 1e6),
                            "longitudeInMicroDegrees": int(longitude * 1e6)
                        },
                        "radius": radius,
                        "radiusUnits": "METERS"
                    }
                },
                "updateMask": "proximity.radius,proximity.geoPoint"
            }
        ]
    }
    facebook_payload = {
        "name": f"Regular Campaign for {location}",
        "targeting": {
            "geo_locations": {
                "custom_locations": [
                    {
                        "latitude": latitude,
                        "longitude": longitude,
                        "radius": radius / 1000,
                        "distance_unit": "kilometer"
                    }
                ]
            }
        },
        "status": "ACTIVE"
    }
    google_response = send_post_request(google_ads_api_url, google_headers,
```

```
google_payload)
    print(f"Google Ads response: {google_response}")
    facebook_response = send_post_request(facebook_ads_api_url,
facebook_headers, facebook_payload)
    print(f"Facebook Ads response: {facebook_response}")
    store_sales_data(location, sales, "regular")

def send_post_request(url, headers, payload):
    response = requests.post(url, headers=headers, data=json.dumps(payload))
    return response.json()

def store_sales_data(location, sales, action):
    try:
        data = {
            "location": location,
            "sales": sales,
            "action": action,
            "timestamp": datetime.now().isoformat()
        }
        file_name = f"{location.replace(',',
'_')}_{datetime.now().strftime('%Y%m%d%H%M%S')}.json"
        s3_client.put_object(
            Bucket=S3_BUCKET_NAME,
            Key=file_name,
            Body=json.dumps(data),
            ContentType='application/json'
        )
    except Exception as e:
        print(f"Error storing sales data: {e}")
```

Documentation used to develop the Lambda function including Google Ads API and Facebook Ads API:

Google Ads API documentation links:

ProximityInfo | Google Ads API

- GeoPointInfo - longitude and latitude microdegrees for campaign location

CampaignCriterion | Google Ads API

- CampaignCriterion | Google Ads API - LocationGroupRadiusUnits meters
- CampaignCriterionOperation | Google Ads API - update, UpdateMask with proximity.radius and proximity.geoPoint

Facebook Marketing API documentation links:
Note: Unfortunately Facebook docs don't allow to link specific points of documentation, so below is a list of 'available fields' and a link to the Basic targeting documentation. When on the API page press Command + F and search for the keyword will help to find these faster.

Basic Targeting - Marketing API

custom_locations:
- custom_locations.latitude
- custom_locations.longitude
- custom_locations.radius - miles or kilometers
- custom_locations.distance_unit

3. Setting up sales visualisation AWS QuickSight

For visualisation, I have considered using Google Data Studio, which requires additional integration with AWS services (such as S3 where we store the sales data), considering our whole solution is in AWS for the sake of time management we will use AWS QuickSight for ease of integration with existing services. Developing this solution in an actual work project I'd integrate the Google Data Studio as it is free to use.

4. Monitoring and maintenance

For monitoring and maintenance purpose we can integrate AWS CloudWatch. We can add logging into the lambda code which would then send notifications, e.g. into a slack channel. This is an idea and would involve research of how this can be implemented.

Logging | Slack

5. Using different cloud providers - GCP

This solution could be implemented using other cloud providers such as Google Cloud Platform. In this project, I have worked out the solution in AWS due to familiarity with the platform. I am eager to learn and work with other platform providers such as GCP. After some research, we'd use these equivalents of AWS services to deploy this solution on GCP.
AWS S3 - Google Cloud Storage
AWS Lambda - Google Cloud Functions
AWS API Gateway - Google Cloud Endpoints
AWS IAM - Google Cloud IAM
AMAZON QuickSight - Google Data Studio

QUESTIONS to provide more clarity and specifications for the solution's development:

1. It is mentioned in the Project specification that the client wants to give additional advertising spending when action is "Boost". This adjustment isn't specified in the CSV. It isn't specified in the requirement either. How is this adjustment done? Is it manual or they'd like us to include that in the automation. If so can the client provide percentage or specific budgets for this so we can include this in the Lambda function and the Google and Facebook payload.
2. Will we have a Google and Facebook tokens and campaign ids from the media activation team? I assume they will set up these campaigns.
3. Does action 'Exclude' mean PAUSING or DELETING the campaing?
4. Is 'Regular' just updating the radius depending whether it is weekday or weekend?
5. Will the email be sent from the same client's email address? If so, what is the address. We can include this in the search query to provide extra security.