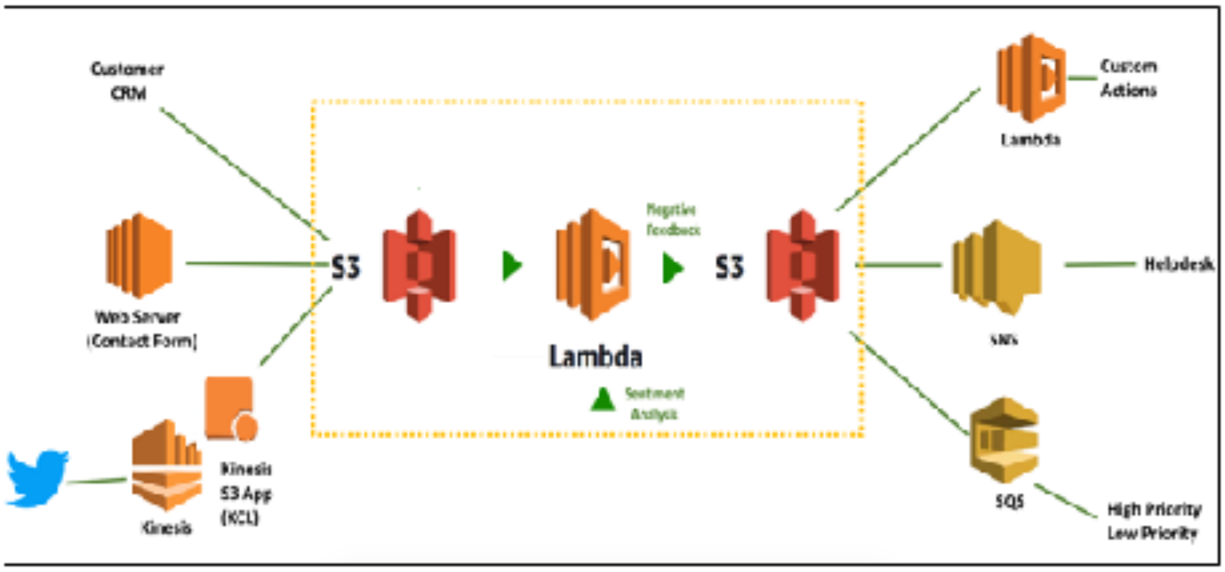
Using Amazon Simple Storage Service (Amazon S3) and AWS Lambda, organizations can collect and prioritize customer feedback based on sentiment analysis, with a serverless architecture to monitor and act upon negative feedback. This solution is based on the following architecture:



How it Works

Customer feedback is formatted as a text file and sent to an S3 bucket. Once an object is stored on S3, a Lambda function will trigger to start a sentiment analysis. If the results from the sentiment analysis are negative, the file is copied to a high-priority S3 bucket. From this high-priority bucket, several things can be done:

- You can have notifications configured with Amazon Simple Notification Service (SNS) to let people know about feedback requiring attention.
- You can trigger another Lambda function to take action or analysis.

Using TextBlob for Natural Language Processing

For the sentiment analysis, we use Python's TextBlob. TextBlob relies on NLTK, an open-source library for processing human language. The library offers lexicon and datasets resources as well as built-in functions for sentiment analysis. The TextBlob library provides APIs for natural language processing (NLP), including a sentiment analysis module with two implementations: PatternAnalyzer and NaiveBayesAnalyzer. In this example, we use PatternAnalyzer.

TextBlob is flexible and allows us to override the default implementation to create our own analyzer and a custom sentiment analysis. Let's get started.

The Lambda function, coded in Python and found within github, will decide if the feedback must be sent to the high-priority bucket.

First, you need to import TextBlob, boto and a few other libraries:

```
from textblob import TextBlob
import json
import urllib.parse
import boto3
```

Then, we will retrieve the file for analysis. We need to urllib the file per the UTF-8 encoding.

```
bucket = event['Records'][0]['s3']['bucket']['name']

key = urllib.parse.unquote_plus(event['Records'][0]['s3']

['object']['key'], encoding='utf-8')

    try:

        response = s3.get_object(Bucket=bucket, Key=key)

        tcontent = response['Body'].read().decode('utf-8')

        print(tcontent)
```

We create the TextBlob item and make a decision based on the polarity of the sentiment analysis. If the sentiment is negative, we download the feedback file and upload it again to the high-priority bucket, using boto S3 functions.

```
blob = TextBlob(tcontent)


        if blob.polarity < 0:

            d_path = '/tmp/{}'.format(key)

            s3.download_file(bucket, key, d_path)

            s3.upload_file(d_path, 'highpriority-customer-

feedback', key)

return "New file in High Priority Folder"
```

Configuring the Environment

First, two buckets need to be created: the regular bucket where customer feedback is stored from multiple channels and the other bucket that will contain the high-priority feedback from our Lambda function.

Bucket name	Access	Region	Date created
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Once created, a Lambda function needs to be defined. As the code has some dependencies on external libraries, a deployment package needs to be uploaded (Node.js). It is important to remember that the .zip file that is part of the deployment code must contain a .py file whose name will be also the name for the handler module.

To build the Lambda function, start by going into the Lambda service in your console and “Create a Function.” Now, you have the option to select “Blueprints Functions” or “Author from scratch.” Select the “Author from scratch” option.

Give a name to your function and an execution role. The function will need to read the customer feedback file that will be stored into S3, so you will need to create a new role based on the pre-defined template “S3 object read-only permissions.”

To be able to write on S3, we need to modify the pre-defined template. The following policy attached to the Lambda role allows us to write to the high-priority S3 bucket. As an example:

```
{
    "Version": "2018-08-05",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::highp-customer-feedback"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::customer-feedback-
omni"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::highp-customer-feedback/*"
            ]
        }
    ]
}
```

Once you click on “Create Function,” the configuration needs to be introduced. Follow the steps and make sure your code includes a “handler” function to pick up the trigger and start the Lambda execution.

Select the .zip file with your deployment package and go to the “Trigger” section. Choose S3 and the bucket that the Lambda trigger will be associated with as well as the type of event, which in this case is “Object created (All).” Leave the checkbox that says “Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger,” as this will save you some set-up time.

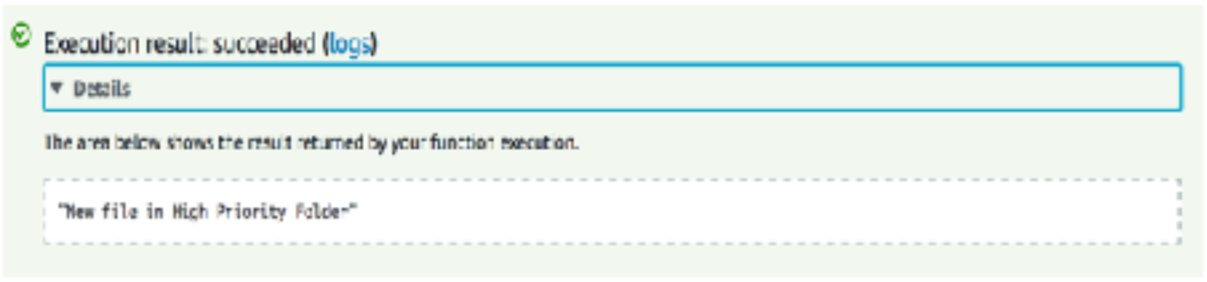
The code needs to know the bucket name it should write to, though the use of environment variables may ease the portability of the code:

```
 d_path = '/tmp/{}'.format(key)
            s3.download_file(bucket, key, d_path)
            s3.upload_file(d_path, 'highp-customer-
feedback', key)
```

Update the code according to the S3 names you created.

Now, when a file containing feedback is uploaded into the source bucket with a complaint, it will be analyzed. If it is negative, it will be copied into the high-priority bucket. Try survey-1.csv and survey-3.csv. Also, find some positive customer feedback to verify that it is not copied into the bucket.

The output of the Lambda function when using the sample complaint will be:



Execution result: succeeded (logs)

▼ Details

The area below shows the result returned by your function execution.

```
"New file in High Priority Folder"
```

You can also find that the file has been automatically copied into the high-priority bucket.

Viewing 1 to 1

	Name	Last modified	Size	Storage class
☐	[illegible]	[illegible]	[illegible]	Standard

Viewing 1 to 1

You can extend this basic functionality with more complex systems and decisions based on the result of the analysis, as well as build your own real-time feedback classification system based on a serverless architecture.