```
!pip install boto3
```

```
Collecting boto3
  Downloading boto3-1.35.3-py3-none-any.whl.metadata (6.6 kB)
Collecting botocore<1.36.0,>=1.35.3 (from boto3)
  Downloading botocore-1.35.3-py3-none-any.whl.metadata (5.7 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3)
  Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3)
  Downloading s3transfer-0.10.2-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.10/dist-packages (from botocore<1.36.0,>=1.35.3->bc
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in /usr/local/lib/python3.10/dist-packages (from botocore<1.36.0,>=1.35.3->bot
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.36.0,>=
Downloading boto3-1.35.3-py3-none-any.whl (139 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.1/139.1 kB 6.2 MB/s eta 0:00:00
Downloading botocore-1.35.3-py3-none-any.whl (12.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.5/12.5 MB 50.0 MB/s eta 0:00:00
Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Downloading s3transfer-0.10.2-py3-none-any.whl (82 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.7/82.7 kB 4.8 MB/s eta 0:00:00
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.35.3 botocore-1.35.3 jmespath-1.0.1 s3transfer-0.10.2
```

```
import boto3
import os
from datetime import datetime, timedelta

os.environ['AWS_ACCESS_KEY_ID'] = 'AKIAQE43JVGOV2N6ROE7'
os.environ['AWS_SECRET_ACCESS_KEY'] = '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'
region_name = 'us-east-1'

lambda_client = boto3.client('lambda', region_name=region_name)
sqs_client = boto3.client('sqs', region_name=region_name)
efs_client = boto3.client('efs', region_name=region_name)
s3_client = boto3.client('s3', region_name=region_name)
cloudwatch_client = boto3.client('cloudwatch', region_name=region_name)

FREE_TIER_LAMBDA_REQUESTS = 1000000
FREE_TIER_LAMBDA_GB_SEC = 400000
FREE_TIER_SQS_REQUESTS = 1000000
FREE_TIER_EFS_STORAGE_GB = 5
FREE_TIER_S3_STORAGE_GB = 5

def fetch_lambda_free_tier():
    response = lambda_client.get_account_settings()
    total_requests = response['AccountUsage']['TotalCodeSize']
    total_memory = 0
    total_cpus = 0

    functions = lambda_client.list_functions()
    for function in functions['Functions']:
        config = lambda_client.get_function_configuration(FunctionName=function['FunctionName'])
        memory_size = config['MemorySize']
        cpu_allocation = memory_size / 1024
        total_memory += memory_size
        total_cpus += cpu_allocation
        total_requests += 1

    free_cpus = total_cpus
    free_memory = total_memory

    return {
        "Total Lambda Requests": total_requests,
        "Free CPUs Used": free_cpus,
        "Free Memory Used (MB)": free_memory
    }

def fetch_sqs_free_tier(queue_name):
    metrics = cloudwatch_client.get_metric_statistics(
        Namespace='AWS/SQS',
        MetricName='NumberOfMessagesSent',
        Dimensions=[{'Name': 'QueueName', 'Value': queue_name}],
        StartTime=(datetime.utcnow() - timedelta(days=30)),
        EndTime=datetime.utcnow(),
        Period=86400,
        Statistics=['Sum']
```

```
        )
        total_messages = sum(datapoint['Sum'] for datapoint in metrics['Datapoints'])

        avg_message_size_kb = 64
        total_message_size_mb = (total_messages * avg_message_size_kb) / 1024  # MB

        free_messages = total_messages if total_messages <= FREE_TIER_SQS_REQUESTS else FREE_TIER_SQS_REQUESTS
        free_message_size = total_message_size_mb if total_messages <= FREE_TIER_SQS_REQUESTS else (FREE_TIER_SQS_REQUESTS * avg_message_size_kb /

        return {
            "Total Messages Sent": total_messages,
            "Free Messages Sent": free_messages,
            "Total Message Size (MB)": total_message_size_mb,
            "Free Message Size (MB)": free_message_size
        }

    def fetch_efs_free_tier():
        file_systems = efs_client.describe_file_systems()
        total_storage = sum(fs['SizeInBytes']['Value'] for fs in file_systems['FileSystems'])
        total_storage_gb = total_storage / (1024 ** 3)  # GB

        free_storage_gb = max(0, FREE_TIER_EFS_STORAGE_GB - total_storage_gb)

        return {
            "Total EFS Storage Used (GB)": total_storage_gb,
            "Free EFS Storage Available (GB)": free_storage_gb
        }

    def fetch_s3_free_tier():
        total_storage = 0
        buckets = s3_client.list_buckets()

        for bucket in buckets['Buckets']:
            metrics = cloudwatch_client.get_metric_statistics(
                Namespace='AWS/S3',
                MetricName='BucketSizeBytes',
                Dimensions=[{'Name': 'BucketName', 'Value': bucket['Name']},
                            {'Name': 'StorageType', 'Value': 'StandardStorage'}],
                StartTime=(datetime.utcnow() - timedelta(days=30)),
                EndTime=datetime.utcnow(),
                Period=86400,
                Statistics=['Average']
            )
            if metrics['Datapoints']:
                total_storage += metrics['Datapoints'][0]['Average']

        total_storage_gb = total_storage / (1024 ** 3)

        free_storage_gb = max(0, FREE_TIER_S3_STORAGE_GB - total_storage_gb)

        return {
            "Total S3 Storage Used (GB)": total_storage_gb,
            "Free S3 Storage Available (GB)": free_storage_gb
        }

    def generate_zero_dollar_plan(num_elements):
        return {
            "Total Processing Hours": num_elements / FREE_TIER_LAMBDA_REQUESTS,
            "Total Cost": 0
        }

    def generate_best_possible_plan(num_elements):
        paid_lambda_requests = max(0, num_elements - FREE_TIER_LAMBDA_REQUESTS)
        cost_lambda = paid_lambda_requests * 0.0000002

        return {
            "Total Processing Hours": num_elements / FREE_TIER_LAMBDA_REQUESTS,
            "Total Cost": cost_lambda
        }

    if __name__ == "__main__":
        num_elements = 1000000

        lambda_metrics = fetch_lambda_free_tier()
        sqs_metrics = fetch_sqs_free_tier("test-topic")
        efs_metrics = fetch_efs_free_tier()
        s3_metrics = fetch_s3_free_tier()
```

```
print(f"Lambda Free Tier Metrics: {lambda_metrics}")
print(f"SQS Free Tier Metrics: {sqs_metrics}")
print(f"EFS Free Tier Metrics: {efs_metrics}")
print(f"S3 Free Tier Metrics: {s3_metrics}")

zero_dollar_plan = generate_zero_dollar_plan(num_elements)
best_possible_plan = generate_best_possible_plan(num_elements)

print("Zero Dollar Plan:", zero_dollar_plan)
print("Best Possible Plan:", best_possible_plan)
```

```
Lambda Free Tier Metrics: {'Total Lambda Requests': 19175507, 'Free CPUs Used': 0.25, 'Free Memory Used (MB)': 256}
SQS Free Tier Metrics: {'Total Messages Sent': 0, 'Free Messages Sent': 0, 'Total Message Size (MB)': 0.0, 'Free Message Size (MB)': 0.0
EFS Free Tier Metrics: {'Total EFS Storage Used (GB)': 0.0, 'Free EFS Storage Available (GB)': 5.0}
S3 Free Tier Metrics: {'Total S3 Storage Used (GB)': 0.0, 'Free S3 Storage Available (GB)': 5.0}
Zero Dollar Plan: {'Total Processing Hours': 1.0, 'Total Cost': 0}
Best Possible Plan: {'Total Processing Hours': 1.0, 'Total Cost': 0.0}
```