}),

Serverless using AWS Lambda for Python Developers
Assignment Solutions

Assignment 1 : Parameters and Return types

```
import json

    import random

    def truck_tracker(event, context):

    if event.latitude and event.longitude:

print('latitude: {}, longitude: {}', event.latitude, event.longitude)

    def get_ticket(event, context):

    if event.payment and float(event.payment) > 0.0:

return {

"statusCode": 200,

"body": json.dumps({

"ticketNumber": random.randint()

}),
}
else:

return json.dumps({

"statusCode": 400,

"error": json.dumps({

"message": "Please provide payment"

})

})
```

Assignment 2 : Create Serverless API

```
template.yml

AWSTemplateFormatVersion: '2010-09-09'

Transform: AWS::Serverless-2016-10-31

Description: >

customer-api
```

```
Globals:

        Function:

        Runtime: python3.8

        Timeout: 30

        Environment:

        Variables:

    CUSTOMERS_TABLE: !Ref CustomersTable

Resources:

        CustomersTable:

        Type: AWS::Serverless::SimpleTable

        Properties:

        PrimaryKey:

    Name: id

    Type: Number

        CreateCustomersFunction:
```

```
        Type: AWS::Serverless::Function
        Properties:
        CodeUri: customers_api/
    Handler: create.lambda_handler
Events:
    CreateCustomers:
    Type: Api
        Properties:
        Path: /customers
    Method: POST
    Policies:
    - DynamoDBCrudPolicy:
        TableName: !Ref CustomersTable
        ReadCustomersFunction:
        Type: AWS::Serverless::Function
        Properties:

        CodeUri: customers_api/

    Handler: read.lambda_handler

    Events:

    ReadCustomers:

    Type: Api

        Properties:

        Path: /customers/{id}

    Method: GET

    Policies:

    - DynamoDBReadPolicy:

    TableName: !Ref CustomersTable

Outputs:

    CreateCustomersAPI:

    Description: "API Gateway endpoint for creating customers"
```

```
    Value: !Sub
"https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaw
s.com/Prod/customers"

    CreateCustomerFunction:

        Description: "Create Customer function ARN"

        Value: !GetAtt CreateCustomerFunction.Arn

        CreateCustomerFunctionIamRole:

        Description: "Create Customer function IAM Role ARN"

        Value: !GetAtt CreateCustomerFunctionRole.Arn
```

create.py

```
        import json

        import boto3

        import os

        dynamodb = boto3.resource('dynamodb')

        table_name = os.environ.get('CUSTOMERS_TABLE')

        def lambda_handler(event, context):
```

```
        customer = json.loads(event['body'])

        table = dynamodb.Table(table_name)

        response = table.put_item(TableName=table_name,
Item=customer)

        print(response)

        return {

    'statusCode': 201,

    'headers': {},

    'body': json.dumps({'message': 'Customer Created'})

}

        read.py

        import simplejson as json

        import boto3

        import os

        from boto3.dynamodb.conditions import Key
        dynamodb = boto3.resource('dynamodb')

        table_name = os.environ.get('CUSTOMERS_TABLE')

        def lambda_handler(event, context):

        table = dynamodb.Table(table_name)

        customer_id = int(event['pathParameters']['id'])

        response =
table.query(KeyConditionExpression=Key('id').eq(customer_id))

    return {

    'statusCode': 200,

    'headers': {},

    'body': json.dumps(response['Items'])

}
```

Assignment 3 : Asynchronous Usecase

```
        import boto3

        import json
```

```
    import os

    s3 = boto3.client('s3')

    sns_client = boto3.client('sns')

    def lambda_handler(event, context):

    topic = os.environ.get('CALCULATED_GRADE_TOPIC')

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

    file_key = event['Records'][0]['s3']['object']['key']

    obj = s3.get_object(Bucket=bucket_name, Key=file_key)

    file_content = obj['Body'].read().decode('utf-8')

    checkout_events = json.loads(file_content)

    for each_event in checkout_events:

if (70 < int(each_event['testScore'])):

score = 'Grade = A'

elif (int(each_event['testScore']) in range(60, 71)):

score = 'Grade = B'
```

```
    elif (int(each_event['testScore']) < 60):

    score = 'Grade = C'

    print(score)

    sns_client.publish(TopicArn=topic, Message=json.dumps(

    {'default': score}), MessageStructure='json')

        def lambda_handler(event, context):

        message = event['Records'][0]['Sns']['Message']

        print(message)

        AWSTemplateFormatVersion: "2010-09-09"

        Transform: AWS::Serverless-2016-10-31

        Description: >

        gradecalculator

Globals:

        Function:

        Timeout: 30
```

```
Resources:

        CalculatedGradeTopic:

        Type: AWS::SNS::Topic

        GradeTrackerBucket:

        Type: AWS::S3::Bucket

        Properties:

        BucketName: !Sub
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}

        GradeCalculatorFunction:

        Type: AWS::Serverless::Function

        Properties:

        CodeUri: grade_calculator/

    Handler: gradecalculator.lambda_handler

    Runtime: python3.9

    Policies:
```

```
  - S3ReadPolicy:
  BucketName: !Sub
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}
  - SNSPublishMessagePolicy:
  TopicName: !GetAtt CalculatedGradeTopic.TopicName
  Environment:
  Variables:
  CALCULATED_GRADE_TOPIC: !Ref CalculatedGradeTopic
  Events:
  S3Event:
  Type: S3
  Properties:
  Bucket: !Ref GradeTrackerBucket
  Events: s3:ObjectCreated:*
  DisplayGradeFunction:
  Type: AWS::Serverless::Function
```

```
Properties:

CodeUri: grade_calculator/

Handler: displaygrade.lambda_handler

Runtime: python3.9

Events:

SNSEvent:

Type: SNS

Properties:

Topic: !Ref CalculatedGradeTopic
```

Assignment 4 : Logging

```
import boto3

    import json

    import os

    import logging
    s3 = boto3.client('s3')

    sns_client = boto3.client('sns')

    logger = logging.getLogger('gradecalculator')

    logger.setLevel(logging.INFO)

    def lambda_handler(event, context):

    topic = os.environ.get('CALCULATED_GRADE_TOPIC')

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

    file_key = event['Records'][0]['s3']['object']['key']

    logger.info('Reading {} from {}'.format(file_key, bucket_name))

    obj = s3.get_object(Bucket=bucket_name, Key=file_key)

    file_content = obj['Body'].read().decode('utf-8')

    checkout_events = json.loads(file_content)

    for each_event in checkout_events:

if (70 < int(each_event['testScore'])):

score = 'Grade = A'
```

```
elif (int(each_event['testScore']) in range(60, 71)):

    score = 'Grade = B'

elif (int(each_event['testScore']) < 60):

score = 'Grade = C'

logger.info('Message being published')

logger.info(score)

sns_client.publish(TopicArn=topic, Message=json.dumps(

    {'default': score}), MessageStructure='json')

    import logging

    logger = logging.getLogger('displaygrade')

    logger.setLevel(logging.INFO)

    def lambda_handler(event, context):

    message = event['Records'][0]['Sns']['Message']

    logger.info('Message being displayed')

    logger.info(message)
    AWSTemplateFormatVersion: "2010-09-09"

    Transform: AWS::Serverless-2016-10-31

Description: >

    gr

    adecalculator

Globals:

    Function:

    Timeout: 30

Resources:

    CalculatedGradeTopic:

    Type: AWS::SNS::Topic

    GradeTrackerBucket:

    Type: AWS::S3::Bucket

    Properties:

    BucketName: !Sub
```

```
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}

        GradeCalculatorFunction:

        Type: AWS::Serverless::Function

        Properties:

        CodeUri: grade_calculator/

    Handler: gradecalculator.lambda_handler

        Runtime: python3.9

    Policies:

    - S3ReadPolicy:

    BucketName: !Sub
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}

    - SNSPublishMessagePolicy:

    TopicName: !GetAtt CalculatedGradeTopic.TopicName

    Environment:

    Variables:

    CALCULATED_GRADE_TOPIC: !Ref CalculatedGradeTopic
```

```
Events:

S3Event:

Type: S3

Properties:

Bucket: !Ref GradeTrackerBucket

    Events: s3:ObjectCreated:*

DisplayGradeFunction:

Type: AWS::Serverless::Function

Properties:

CodeUri: grade_calculator/

Handler: displaygrade.lambda_handler

Runtime: python3.9

Events:

SNSEvent:

Type: SNS
```

```
    Properties:

    Topic: !Ref CalculatedGradeTopic
```

Assignment 5 : Error Handling

```
        AWSTemplateFormatVersion: "2010-09-09"

        Transform: AWS::Serverless-2016-10-31

        Description: >

        gradecalculator

Globals:
        Function:

        Timeout: 30

Resources:

        GradeCalculatorDLQ:

        Type: AWS::SNS::Topic

        CalculatedGradeTopic:
```

```
    Type: AWS::SNS::Topic

    GradeTrackerBucket:

    Type: AWS::S3::Bucket

    Properties:

    BucketName: !Sub
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}

    GradeCalculatorFunction:

    Type: AWS::Serverless::Function

  Properties:

    CodeUri: grade_calculator/

  Handler: gradecalculator.lambda_handler

  Runtime: python3.9

  DeadLetterQueue:

  Type: SNS

  TargetArn: !Ref GradeCalculatorDLQ

  Policies:
  - S3ReadPolicy:

    BucketName: !Sub
${AWS::StackName}-${AWS::AccountId}-${AWS::Region}

  - SNSPublishMessagePolicy:

  TopicName: !GetAtt CalculatedGradeTopic.TopicName

  Environment:

  Variables:

  CALCULATED_GRADE_TOPIC: !Ref CalculatedGradeTopic

  Events:

  S3Event:

  Type: S3

  Properties:

    Bucket: !Ref GradeTrackerBucket

    Events: s3:ObjectCreated:*

    DisplayGradeFunction:
```

```
    Type: AWS::Serverless::Function

    Properties:

    CodeUri: grade_calculator/

    Handler: displaygrade.lambda_handler

    Runtime: python3.9

Events:

SNSEvent:

Type: SNS

Properties:

Topic: !Ref CalculatedGradeTopic

    ErrorHandlerFunction:

    Type: AWS::Serverless::Function

    Properties:

    CodeUri: grade_calculator/

    Handler: errorhandler.lambda_handler
    Runtime: python3.9

Events:

SNSEvent:

Type: SNS

Properties:

Topic: !Ref GradeCalculatorDLQ

    import logging

    logger = logging.getLogger('displaygrade')

    logger.setLevel(logging.INFO)

    def lambda_handler(event, context):

    message = event['Records'][0]['Sns']['Message']

    logger.info('Message being displayed')

    logger.info(message)
```