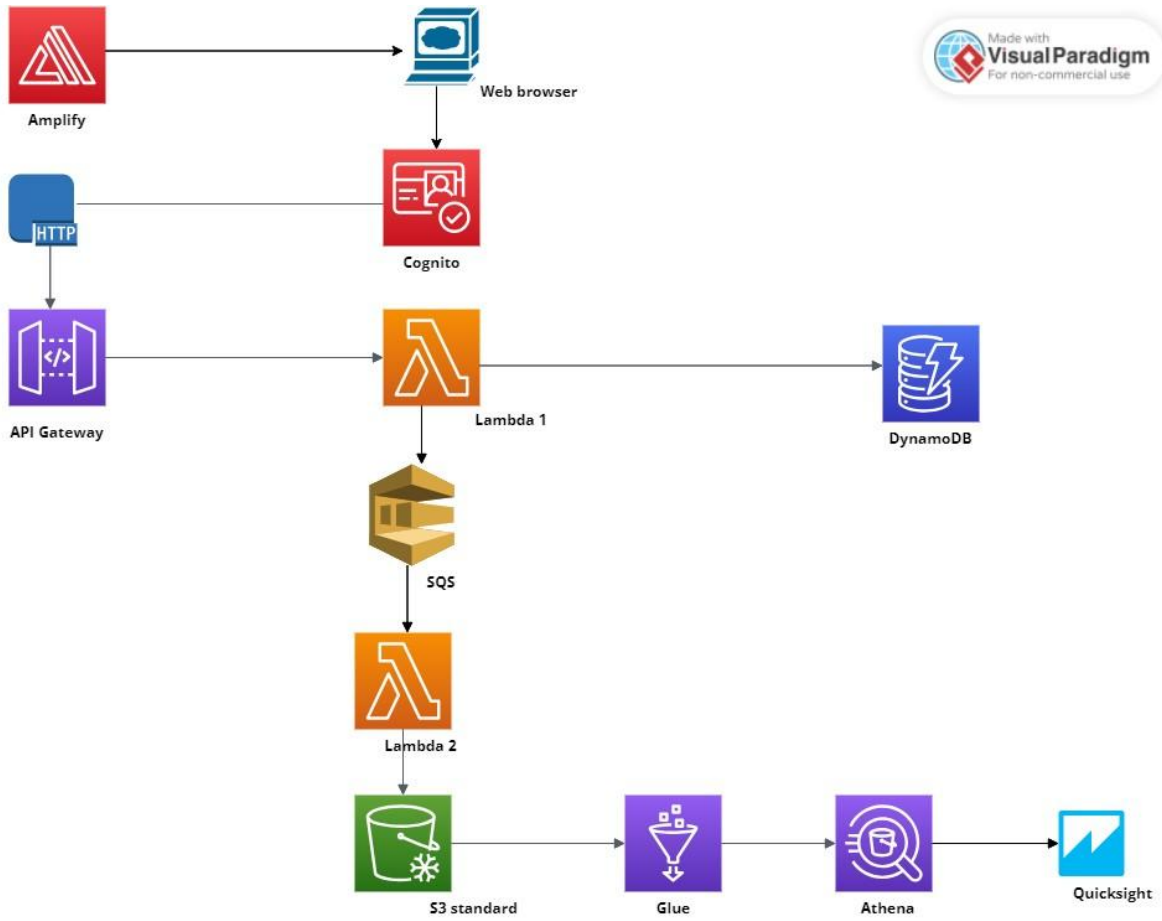
AWS IoT Engineer (no-code):

1. Diseña la arquitectura para una API con servicios serverless (lambda, API GW, Cognito, etc…), librerías que utilizarías, etc…



Explicación de cada componente:

AWS Amplify sirve ficheros HTML, CSS y JavaScript que se cargan en el navegador web del usuario, en ese frontend el usuario se autentica a través del servicio Cognito. Al ser una web estática también se podían haber alojado los ficheros de la web en S3.

A continuación se dirige una petición HTTP que es recogida por la API Gateway que sirve de trigger para Lambda 1 que recoge y procesa el input, guarda el resultado en dynamoDB y envía un mensaje a través de la SQS.

La SQS es el trigger que activa Lambda 2 que se ocupa de leer el histórico en dynamoDB y generar una comparativa, cuyos resultados generan por cada ejecución un fichero que se guarda en S3.

A continuación mediante una sencilla ETL con Glue se curan, transforman y catalogan los datos de S3 y se visualizan en Athena y a partir de ahí se elabora el correspondiente informe en Quicksight.

En cuanto a librerías, se utilizaría boto3 para acceder al cliente de AWS, json para el formateo y pandas o spark para manejar los datos en la ETL, dependiendo del volumen.

2. Describe las diferencias entre un topic y una cola en stream processing

Un topic sigue el modelo publicación-subscripción a través de un agente, como por ejemplo Apache Kafka, Amazon MSK en AWS ó Amazon MQ para otras implementaciones. En estos modelos el agente publica mensajes en los topic que tenga asignados, y el receptor debe suscribirse al topic para poder recibir los mensajes.

El topic es el canal por el que se transmiten los mensajes, para entenderlo se puede considerar la analogía de un canal de TV o una emisora de radio que se sintoniza para recibir el mensaje.

En una cola, el modelo de comunicación es mucho más simple y se basa en la comunicación punto a punto, toda la configuración subyacente es transparente, el desarrollador simplemente debe instanciar el identificador de la cola en el recurso emisor y receptor. El ejemplo más claro en AWS es configurar una cola SQS como trigger de una función lambda.

3. ¿Para qué se utiliza un Load Balancer? ¿Qué tipos hay?

Un Load Balancer permite distribuir el tráfico en varios servidores, (instancias de EC2, contenedores etc.) con esto se consiguen múltiples beneficios:

- Mejora la disponibilidad, al di sminuir la latencia por la mayor proximidad geográfica
- Mejora la escalabilidad, por la distribución de la carga computacional en varios servidores
- Mejora de la seguridad, la distribución de las peticiones previene ataques DDos
- Sustancial mejora del rendimiento, debido a la escalabilidad horizontal de recursos computacionales
- Permite eliminar servidores inestables

Tipos de Load Balancer:

- Application Load Balancer: sirve para manejar tráfico HTTP y HTTPS. Permite el tráfico a diferentes servicios según la URL de la petición.
- Network Load Balancer: optimizado para manejar tráfico TCP. Ofrece rendimiento y baja latencia.
- Classic Load Balancer: un balanceador de carga más antiguo que puede manejar tanto tráfico HTTP/HTTPS como TCP.
- Gateway Load Balancer: permite la implementación y gestión de dispositivos virtuales, como firewalls y sistemas de detección de intrusos.

4. Diferencias entre servicios serverless e IaaS, con ejemplos.

El concepto serverless hace referencia a recursos que permiten ejecutar código de forma directa, sin la necesidad de administrar la infraestructura subyacente, el propio proveedor del cloud se ocupa de todo lo necesario, para que el desarrollador se centre única y exclusivamente en el código. El ejemplo por excelencia es lambda para pequeños scripts o Fargate para contenedores.

En el modelo IaaS, (Infrastructure as a service) el proveedor cloud facilita al usuario únicamente los recursos computacionales, siendo responsabilidad del cliente configurar el entorno, instalar las dependencias, librerías, sistema operativo etc., en este modelo el proveedor cloud únicamente es responsable del hardware y de la infraestructura física necesaria, (servidores, centros de datos, energía etc.). Son ejemplos claros de este concepto los servicios EC2 y EMR.

Preguntas AWS IoT Engineer (code):

1. ¿Cómo crearías una regla en AWS IoT que envíe un correo electrónico cuando se detecta una anomalía en los datos?. Utiliza el lenguaje de programación, cli o framework que prefieras.

Es bastante simple, se puede crear una rule con el siguiente comando en AWS CLI:

```
{
    "sql": "SELECT * FROM 'iot/inputData' WHERE 'temperature' > 40",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
        {
            "sns": {
                "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
                "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
                "message": {
                    "default": "Temperature alert heat wave with temperature: ${temperature}"
                }
            }
        }
    ]
}
```

Este comando desencadena la acción de enviar un mensaje a través de la SNS my-sns-topic a los usuarios subscriptores de la misma cuando el valor temperature recogido supere el valor de 40.

Hay que tener en creado el topic SNS previamente, bien a través de la consola de AWS o mediante template de CloudFormation.

2. ¿Cómo harías una función Lambda que procese los datos de un dispositivo IoT y los almacene en DynamoDB?. Utiliza el lenguaje de programación, cli o framework que prefieras.

Se debe crear el la regla para AWS IoT, para ello se debe ejecutar en AWS CLI el siguiente comando:

```
aws iot create-topic-rule --rule-name inputData --topic-rule-payload file://inputData.json
```

Se puede proceder a crear la función lambda como tal, bien a través de la consola de AWS o mediante template de CloudFormation.

A continuación se deben asignar permisos al rol que utilizará la lambda:

```
aws lambda add-permission --function-name write-dynamo --region eu-west-1 --principal iot.amazonaws.com --source-arn arn:aws:iot:region:account-id:rule/inputData --source-account account-id --statement-id unique_id --action "lambda:InvokeFunction"
```

Una vez asignados los permisos a la lambda, se programará el script a ejecutar, que a grandes rasgos podría ser un ejemplo parecido al siguiente:

```
import json
import boto3
from datetime import datetime as dt


def lambda_handler(event, context):
    ''''
    Expected event format:
    {
        body:  {
            temperature: 21.2,
            humidity: 42,
            sector: north
```

```
        }
    }
    '''

    def assign_fire_risk(item):
        '''
        Calcule fire risk depends of temperature
        :param item: json input event
        :return: json item with fire-risk field
        '''

        if item['temperature'] >= 30:
            fireRisk = 'high'
        elif item['temperature'] >= 20:
            fireRisk = 'medium'
        else:
            fireRisk = 'low'
        return fireRisk

    # Create DynamoDB resource
    dynamodb = boto3.resource('dynamodb')

    # DynamoDB table
    table = dynamodb.Table('climatology')

    # Get data from event and add timestamp
    item = json.loads(event['body'])
    item['timestamp_update'] = dt.now().isoformat()

    # Get fire risk depends of temperature
    item['fire-risk'] = assign_fire_risk(item)

    # Insert item in climatology table of dynamoDB
    response = table.put_item(Item=item)

    # Logging response in cloudWatch
    if response['ResponseMetadata']['HTTPStatusCode'] == 200:
        print(f"Item inserted successfully: {item}")
    else:
        print(f"Error inserting item: {response['Error']}")

    end_timestamp = dt.now()
    print(f'End of execution: {end_timestamp}')
```

Es un simple ejemplo que lee el evento de entrada, una captación de datos climatológicos, añade los campos fecha y riesgo de incendio e inserta el ítem en formato json en la tabla climatology de dynamoDB, se utilizan como librerías boto3 para acceder al SDK de AWS, json para formatear el json y datetime para introducir la fecha.

En cualquier caso, de no necesitar procesar los datos, hay opciones para crear IoT rule desde AWS CLI para directamente insertar los datos en dynamoDB sin necesidad de una lambda.

3. Escribe en cloudformation la definición de una función lambda que pueda escribir en el bucket de s3 “my-bucket”, solamente en el prefijo “data/”

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "",
  "Resources" : {
```

```
    "LambdaWriteS3": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "FunctionName": "LambdaWriteS3",
        "Handler": "index.handler",
        "Role": { "Fn::GetAtt": ["RoleLambdaS3", "Arn"] },
        "Code": {
          "S3Bucket": " my-bucket",
          "S3Key": "s3-write-lambda-code.zip"
        },
        "Runtime": "python3.8"
      }
    },
    "RoleLambdaS3": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "lambda.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "Policies": [
          {
            "PolicyName": "S3WritePolicy",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": "s3:PutObject",
                  "Resource": "arn:aws:s3:::my-bucket/data/"
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```

Esta plantilla define una lambda con código python que basándose en el principio del mínimo privilegio en AWS IAM, únicamente tiene permiso para la acción declarada en la política asociada al rol RoleLambdaS3 que es escritura en el prefijo de S3 ***/data*** del bucket ***my-bucket***

Si se quisiera otorgar permisos de escritura a niveles inferiores de la ruta ***/data***, bastaría con modificar la línea Resource por lo siguiente: `"Resource": "arn:aws:s3:::my-bucket/data/*"`

4. ¿En qué se basa la facturación de las funciones lambda. ¿Para qué tipo de operaciones recomendarías su uso y para cuales no?

La facturación de las funciones lambda depende principalmente de tres variables:

- Tiempo de ejecución en milisegundos
- Volumen de memoria asignada, memoria RAM o de procesamiento.
- Número de ejecuciones, se disponen de 1 millón de peticiones gratuitas al mes si se seleccionan arquitecturas basadas en x86 y Arm

Las funciones lambda son indicadas para trabajos ligeros desencadenados por eventos. Hay que tener en cuenta que hay un tope de memoria y un timeout de 15 minutos por tanto deben ser pequeños scripts, transformaciones de datos poco exigentes en recursos computacionales o inserciones en base de datos unitarias. La potencia de AWS lambda proviene de la posibilidad de contar con múltiples ejecuciones simultáneas del mismo proceso, no de la capacidad unitaria del proceso en sí mismo.

No es recomendable para tareas pesadas en memoria o en tiempo que requieran un elevado aprovisionamiento de recursos como por ejemplo ETLs o tareas propias de Big Data & ML, para eso están otra serie de recursos como GlueJobs, Fargate, SageMaker o EC2, EMR, etc.