Assignment 3

ICE191 Software Architecture / Cloud Computing
Ximena Lizeth González Plascencia 32438

1. **Create a CloudFront distribution for your website and explain each step with great technical detail. 50 points.**

CloudFront is a content delivery network (CDN) that aids in the quick delivery of static and dynamic content such as web pages, images, videos and APIs. For starters, a CDN is a large amount of servers distributed around the world, their purpose is that of caching content.

These caching servers store contents from other servers, known as origin servers, and the network's system routes requests from clients to the caching server closest to them. This system improves delivery times since the client request doesn't have to travel all the way to the origin server to retrieve the content. Additionally, if the caching server doesn't have the requested content, the network will route the client's request to the origin server and send a copy of the contents to the server that didn't have them, where they'll be stored in cache ready for a future request. This is the whole purpose of a CDN, reducing latency and improve content delivery speed,

AWS CloudFront network is structured through edge locations around the world that cache content from origin servers (in this case the origin server being a S3 bucket), and once a user makes a request, it serves the content from the server closest to the user. CloudFront (or any CDN) works great with static content, such as static website in a S3 bucket, because this type of content rarely needs updates and remains the same to all users.

Now, a CloudFront distribution is a set of configurations settings to describe the way the CloudFront service is to distribute some content. It's a pointer to the original content (S3 bucket). The origin of the content has to be specified in the distribution not only so that CloudFront distributes it among edge locations, but also to know where to get the content in case a request comes in and the content isn't stored in the cache of the edge locations.

To create a distribution for an S3 bucket, use the following command

```
                        aws cloudfront create-distribution \
--origin-domain-name ximena.cetystijuana.com.s3-website-us-east-1.amazonaws.com \
                    --default-root-object index.html
```

- `aws cloudfront` is the Amazon CloudFront API Reference, this is how through the cli the user can communicate with the service through commands.

- `create-distribution` indicates the creation of a CloudFront distribution, this can be viewed as a JSON model that specifies attributes regarding the way the content will be distributed and delivered.

- `--origin-domain-name` is the flag used to indicate the domain name of the origin of the content, in this case the endpoint URL of the S3 bucket.

- --default-root-object index.html makes CloudFront request the bucket's index.html file every time a request for the root URL comes in, by doing this, exposing the contents of the distribution is avoided.

This command creates a distribution with default values, the response once having run the command is all these values in a JSON format, as seen in the image:

In order to inspect more in depth what the distribution configurations are and to store the distribution, save it in a JSON file by running the following command:

```
aws cloudfront get-distribution \
--id E3O5AQD3SP4CE9 > dist-config.json
```

- `get-distribution` is the command to get information about a certain distribution, the output is in a JSON format with all the elements that make up a distribution.

- `--id` is the flag to indicate the distribution's ID ("Id" attribute, inside the "Distribution" section), not to be confused with the Etag.

- By running `> dist-config.json` indicates that the output of the command is to be saved in the `dist-config.json` file

The following are some of the key parts that can be seen inside the JSON file:

- Origins: this the instance from which CloudFront gets the content from, in this case the origin is an S3 bucket.

```
"Origins": {
    "Quantity": 1,
    "Items": [
        {
            "Id": "ximena.cetystijuana.com.s3-website-us-east-1.amazonaws.com-1675744554-45590",
            "DomainName": "ximena.cetystijuana.com.s3-website-us-east-1.amazonaws.com",
            "OriginPath": "",
            "CustomHeaders": {
                "Quantity": 0
            },
```

  - Examples of more in depth configurations in this section include:
    - *Origin Access Identity*, which is a way of only allowing requests to the bucket that come from CloudFront.
    - Signed URL and/or cookies (URL/cookie that includes additional information, such as an expiration date and time, that gives more control over access to content) that CloudFront can use it to access the content only if the content of the URL/cookie matches wha is established in the configuration.

- Behaviors: this where most of the configurations are and there is a large amount of them, but there's always one default Behavior.
  - Examples of what can be done in behaviors are:
    - Enforcing policies.
    - Change the type of content being delivered depending on the source of the request.
    - Cache behaviors such as how long or what objects will stay in cache.

```
"DefaultCacheBehavior": {
    "TargetOriginId": "ximena.cetystijuana.com.s3-website-us-east-1.amazonaws.com-1675744554-45590",
    "TrustedSigners": {
        "Enabled": false,
        "Quantity": 0
    },
    "TrustedKeyGroups": {
        "Enabled": false,
        "Quantity": 0
    },
    "ViewerProtocolPolicy": "allow-all",
    "AllowedMethods": {
        "Quantity": 2,
        "Items": [
            "HEAD",
            "GET"
        ],
        "CachedMethods": {
            "Quantity": 2,
            "Items": [
                "HEAD",
                "GET"
            ]
        }
```

- Restrictions, Errors, Logs and Tags
    - Examples include:
        - Restricting access to content depending on the geographical location of the request.
        - Throwing an error page in case of an error in the request (also setting what response code to be sent, like for example a “404 Not Found”).
        - There’s an option to log all incoming requests.

- Security: implement secure protocols such a HTTPS, SSL/TLS encryptions, SSL certificates when delivering the content.

- Example of extra AWS features:
    - Origin Shield: helps to minimize the origin’s request load.
    - Lambda@Edge: allows to execute code in the cloud that can customize the content being delivered to clients.

An important element to note is the distribution’s domain name *d1imvz12keorpi.cloudfront.net*. Now a distribution that has the S3 bucket ximena.cetystijuana.com.s3-website-us-east-1.amazonaws.com is created, and now CloudFront will send the distribution’s configuration to all of its edge locations. Finally, the contents of the bucket can be accessed through the domain name d1imvz12keorpi.cloudfront.net.

2. **Update your DNS Record to route to the CloudFront end point and explain each step with great technical detail. 20 points.**

In the `record.json` file make the following changes:

```
{
  "Comment": "CREATE record ",
  "Changes": [{
  "Action": "UPSERT",
              "ResourceRecordSet": {
                          "Name": "ximena.cetystijuana.com",
                          "Type": "CNAME",
                          "TTL": 300,
```

```
                    "ResourceRecords": [{ "Value":
"d1imvz12keorpi.cloudfront.net"}]
                }
    }]
}
```

- The *Action* element is changed to UPSERT since the DNS record already exists, its values are being updated.

- The *ResourceRecords* element is changed so that instead of routing to the S3 bucket, it routes to the CloudFront end point. This value is obtained in the distribution's DomainName element in its JSON model.
    - In this case using the domain name CloudFront assigned when creating the distribution: d1imvz12keorpi.cloudfront.net.

Then use the route53 cli to run the following command and have the changes made in the DNS record.

```
aws route53 change-resource-record-sets \
--hosted-zone-id Z03346142C3RKH191036Y \
    --change-batch file://record.json
```

- The --hosted-zone-id is obtained by executing `aws route53 list-hosted-zones`, this returns information about the hosted zone associated with the AWS accounts.

```
{
    "HostedZones": [
        {
            "Id": "/hostedzone/Z03346142C3RKH191036Y",
            "Name": "cetystijuana.com.",
            "CallerReference": "RISWorkflow-RD:fc4e3528-
b645-4b45-8c60-ea43b3e4363f",
            "Config": {
                "Comment": "HostedZone created by Route53
Registrar",
                "PrivateZone": false
            },
            "ResourceRecordSetCount": 32
        }
    ]
}
```

- The --change-batch flag is the path to the JSON file where the modified DNS record is located at. This outputs a JSON model that contains information about changes made to the hosted zone.

The output is as follows:

```
{
    "ChangeInfo": {
        "Id": "/change/C00392303B1K26XTUREW",
        "Status": "PENDING",
        "SubmittedAt": "2023-02-09T03:28:09.221000+00:00",
        "Comment": "UPSERT record "
    }
}
```

- The status of the change is returned as *PENDING* because when a change is submitted, Route 53 has to spread the made changes to all of its authoritative DNS servers.
    - Authoritative DNS servers store the most recent and accurate information (DNS records) for a domain (it stores lists of domain names and their associated IP addresses).
        - Analogy: they're like the phone book company that publishes multiple phone books, one per region. They have copy of the regional phone book that matches IP addresses with domain names and there are different authoritative DNS servers that cover different regions. It responds to requests from a recursive DNS server (i.e. the person who needs to look up a number) about the correct IP address assigned to a domain name.
    - When this spreading is done, the status is changed to *INSYNC*

Now we can see that `ximena.cetystijuana.com` now points to the CloudFront distribution's domain.

By doing these changes the delivery of the static web site is improved in speed and reduced in latency since now the contents of the bucket are distributed among edge locations and CloudFront can route the requests for the website to the edge location closest to the client.

Requests that go to ximena.cetystijuana.com will now be managed in CloudFront in the way that it will route requests to the edge location closest to the user and serve the cached content from there.

EXTRA:

- Due to a lot of traffic on the domains under cetystijuana.com, a wildcard ACM certificate had to be added to the distribution.
    - An ACM (AWS Certificate Manager) certifcate handles the complexity of provisioning, managing, and deploying public and private SSL/TLS certificates for AWS websites and applications.
    - TLS/SSL certificates are a way to secure internet connections by encrypting data sent between a browser, the website a user is visiting, and the website server.

- A certificate can be created by running the following command:

```
request-certificate –domain-name *.cetystijuana.com
```

- This returns a certificate for *.cetystijuana.com, therefore any subdomain of cetystijuana.com can use it.
    - The certificate is identified by its ARN (Amazon Resource Names) string (output of the command).

- In the DistributionConfig section of the distribution the following has to be modified in order to include the certification in the CloudFront distribution:

```
"PriceClass": "PriceClass_All",
    "Enabled": true,
    "ViewerCertificate": {
        "CloudFrontDefaultCertificate": false,
        "ACMCertificateArn": "arn:aws:acm:us-east-1:292274580527:certificate/
e961006c-fa16-48ce-aeae-1f093f83db26",
        "Certificate": "arn:aws:acm:us-east-1:292274580527:certificate/
e961006c-fa16-48ce-aeae-1f093f83db26",
        "SSLSupportMethod": "vip",
        "MinimumProtocolVersion": "TLSv1",
        "CertificateSource": "acm"
    }
```

- This section establishes that the distribution isn't using CloudFront's default certificate, and is instead using the one created for cetystijuana.com. The certificate's ARN is included and specification that it's an ACM certificate, the SSLSupportMethod that indicates the distribution accepts HTTPS connections from all viewers and MinimumProtocolVersion which specifies the minimum SSL/TLS protocol the distribution is to use when communicating with users.

- The changes are made in the distribution by running the following command:

```
aws cloudfront update-distribution --id E3O5AQD3SP4CE9 \
--distribution-config file://update-dist.json \
--if-match E3U4HHRCVVO21H
```

    - The aforementioned changes are made in a separate JSON file (note: only the DistributionConfig is included in this file, not the whole distribution JSON file).
    - The distribution's id is included, the path to the JSON with the updates and the flag —if-match with the Etag of the most recent version of the distribution (this is specific to updating web content, this is to ensure the most recent version is being updated and no older versions are considered as official).

- The output of this command is the distribution's whole JSON model, including the new changes.

3. **Explain what it means to minify website resources (html, js, css) and the advantages and disadvantages of this process. 10 points.**

Definition:

- Removing unnecessary/redundant data or simplifying code from resources without changing how the resources are processed by the web browser (i.e. changing or deleting data without affecting the functionality of the resource).
    - Reduced or removed data can include comments, unused code, line breaks, variable/function names and white spaces.

- Its useful since the browser reads all extra characters found in the website's resources, making it heavier and larger to load.

- A typical CDN offers automated minification, the original, uncompressed files remain in main server, while the CDN automatically stores minified variants on its caching servers.

Advantages

- Makes code lighter which in return increases access speed and saves memory.
    - Since it's lighter it takes less time to render, and as consequence the resources can be distributed quicker to users (loads faster).
- Reduces bandwidth usage from the user's side.
- A faster website can earn a high ranking by a web a browser given its speed.
    - For example, Google rewards fast websites with higher rankings (by giving them more visibility).

Disadvantages

- Some minified elements are rendered differently in some web browser, affecting the intended functionality of the original resource (specially when minification is applied to HTML).
- Minification can remove maintainability/readability.
- The process isn't very practical when it comes to very large file, either it be by human error or a mistake done through an automated tool, minification can lead to errors that because of this process are hard to debug (since all variables and comments have been removed).

4. **Write a python script (and explain each step with great technical detail) to (20 points)**
    a. **Create or update a record in the Students DynamoDB table based on the id.**

The first step is to import boto3, which is a python library used for interacting with AWS services, such a DynamoDB. It's formerly know as an SDK (Software Development Kit), which is a set of tools that help build software for a certain platform, in this case the platform being AWS.

Once boto3 is imported, the next step is connecting boto3 to the Students DynamoDB instance, boto3 has access to the configuration and credentials files, so connecting to the Students table is as simple as initializing a dynamodb instance (instance of Boto3's DynamoDB.ServiceResource class) with the ***boto3.resource*** method (used for identifying the type of service to be interacted with), in this case representing Amazon DynamoDB (another example would be S3). To interact with a specific table, Boto3's DynamoDB.Table method can be used, which specifies that the resource is a DynamoDB table (this doesn't create a table, only instantiates it), indicated by its name as a string (in this case table_name = "Students:):

```
def initiate_dynamodb_table(table_name: str):
        dynamodb = boto3.resource("dynamodb")
        table = dynamodb.Table(table_name)
        return table
```

To update an entry in the table, use the following method (Boto3's way of running the command *aws dynamodb update-item*; communication with the API):

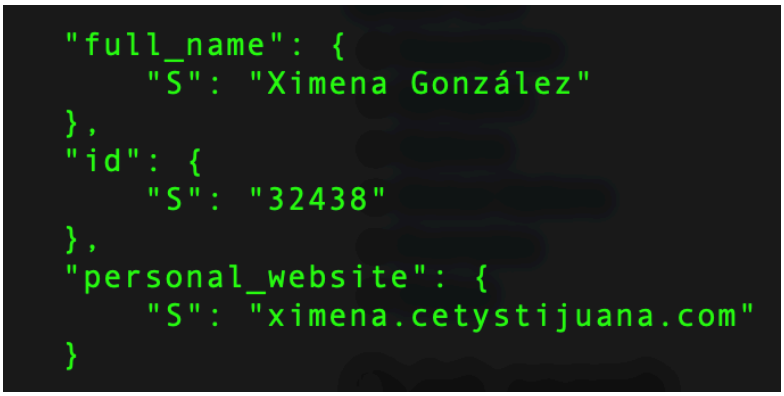
```
def update_table(table: TableResource, key: dict, expression: str, expression_attr: dict):
    table.update_item(Key=key,
                      UpdateExpression=expression,
                      ExpressionAttributeValues=expression_attr
                     )
```

The update_item method allows to edit an existing record, it takes the **Key** argument, which is a dictionary that has primary key of the item to retrieve (in this case key = {"id": "32438"}).

The UpdateExpression argument is a string (similar to a query; the sintaxis comes from dynamodb's documentation) that declares what is to be done (in this case expression = "SET full_name = :val1"), in this case SET replaces the old value with whatever value val1 has.

The ExpressionAttributeNames argument is a dictionary with the key-value pair a strings, the key being the variable name used in the expression and the value being the new value the record's attribute is going to take (in this case expression_attr = {":val1": "Ximena González"}).

This in return makes the changes, confirmed by running *aws dynamodb scan --table-name Students:*



b. Delete a record in the Students DynamoDB table based on the id

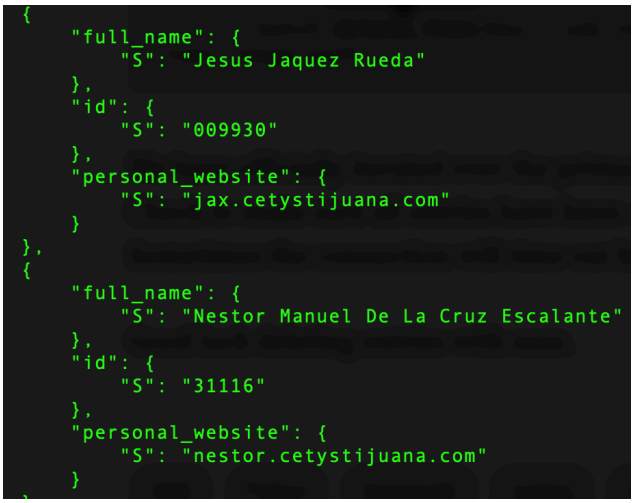
To delete an entry in the table, run the following command (Boto3's way of running the command *aws dynamodb delete-item*), same as the last question, it takes the **Key** argument, which is a dictionary that has primary key of the item to retrieve (in this case key = {"id": "32438"}):

```
def delete_from_table(table: TableResource, key: dict):
    table.delete_item(Key=key)
```

This in return makes the changes, confirmed by running *aws dynamodb scan --table-name Students:*

Before deleting the item:

After deleting the item:



c. Find a record in the Students DynamoDB table by id.

To locate a specific record by id, the *get_item* method can be used. It takes the **Key** argument, which is a dictionary that has primary key of the item to retrieve (in this case key = {"id": “32438"}).

```
def get_from_table(table: TableResource, key: dict):
    response = table.get_item(Key=key)
    print(response)
    return response
```

The response is a dictionary that contains information about the entry (attributes and their value), and metadata (id, date, content format, etc.) about the response (Boto3’s way of running the command *aws dynamodb get-item*):

```
{'Item': {
       'full_name': 'Ximena Lizeth González Plascencia’,
       'id': ‘32438',
       'personal_website': 'ximena.cetystijuana.com'},

'ResponseMetadata': {
       'RequestId': '29M57I43JI5O6MHMI66AIBK6ABVV4KQNSO5AEMVJF66Q9ASUAAJG',
       'HTTPStatusCode': 200,

'HTTPHeaders': {
       'server': 'Server',
       'date': 'Sat, 11 Feb 2023 02:02:20 GMT',
       'content-type': 'application/x-amz-json-1.0',
       'content-length': '135',
       'connection': 'keep-alive',
       'x-amzn-requestid': '29M57I43JI5O6MHMI66AIBK6ABVV4KQNSO5AEMVJF66Q9ASUAAJG',
       'x-amz-crc32': '1217378473'},

'RetryAttempts': 0}}
```

NOTE:

- In order to delete the it item with "id": “32438” (the one deleted in step b), it had to be created and inserted in the table, using the following function, where Item is a dictionary with all the attribute and their values to be inserted (in this case item = {'id': ‘32438', "full_name": "Ximena Lizeth González Plascencia”, “personal_website": "ximena.cetystijuana.com"})

```
def insert_to_table(table: TableResource, item: dict):
    table.put_item(Item=item)
```

5. **Explain the difference between Growth mindset and Fixed mindset according to Carol Dweck 10 points.**

The growth mindset is the ability to view everything as an opportunity and not being afraid of making mistakes. Obstacles are taken on as a learning opportunity, not to be feared to the point of avoidance. People with this kind of mindset don't view or measure their success by metrics or objects, instead their success is what they learned through their journey to achieving something.

The fixed mindset is the view that everything must be perfect done the very first time, that it's better to go on the easy route rather than taking on challenges due to being afraid of making mistakes. Obstacles are the limit, the moment it gets hard, the effort ceases. People with this kind of mindset are static, they believe their skills and intelligence stay the same.

The main difference is that the fixed mindset is very binary, either you have or you don't. There's no room for improvement, the only option is excellence. On the other hand, the growth mindset is all about perseverance, there's always a way to figure it out and achieve set goals, even if it takes a few stumbles and hardships. Like Dr. Dweck stated, in a growth mindset failure just implies a "not yet". Unlike in a fixed mindset where failure is seen as a "stop here".