Assignment 4

ICE191 Software Architecture / Cloud Computing

1. **Describe the concept of throttling in APIs. 15 points.**
   - Definition
     - Throttling in APIs refer to limiting number of API request a user can make in certain period. First able we need to understand the meaning of APIs, basically this tool is a gateway between a user and a software application.
   - Example
     - One great example of this in a website could be a website that gather data about movies. Your API provider has implemented throttling to limit the number of requests per minute to 10, why this limited of request the API will start rejecting additional request until the next minute and this mean that our application while being throttled, it will be prevented from receiving any data for the duration of the throttling period.
     - That means that you only can support or give attention to this request in the time that you selected.
   - Reasons to use Throttling API.
     - Protecting the API server
       - Throttling can help to prevent the API server from becoming overwhelmed with the requests that it receives which could lead to performance issues in the application.
       - A prevention to the common attackers attempts to overload the server with traffic from multiple sources called "distributes denial of services" DDoS, with this tool it helps to ensure fair usage of the API.
       - This tool can help to ensure that the API can scale to handle increases demand by limiting the number of requests that can be made.
   - Amazon API Gateway give us four basics of throttling-related settings.
     If you want to apply throttle its important how to use this tool by Amazon API Gateway.

  - AWS throttling limits, like we said before, a limit its stablished to prevent overwhelming in your API.
  - Per-account limits are applied to all APIs in an account in specific Region. So, the account-level stablished before can be increased upon request.
  - Per-API, per-stage throttling limits are applied at the API method level for a stage.
  - Per-client throttling limits are applied to clients that use API keys associated with your usage plan as client identified
- To conclude with this concept is crucial tool for managing the performance, availability, and of course security of your API. So, with all this you can use your APIs more efficiently and effectively.

Me hace falta agregar ejemplos y como se utiliza

2. **Describe the concept of pagination in APIs. 15 points.**
   - Meaning
     - In our life we have seen it, but never think about it, a clear example is when you do a google research and after that you see that, in the page bottom you don't see more result only the more results button that's in practice what a pagination does in a website.

Additionally the meaning of this is the practice of breaking up large sets of data into smaller, so with that you have more manageable blocks or pages that can be requested by clients through the API, this happened when API returns a large amount of data in response to a request, it can be challenging for the client to process all of the data at once, that's why pagination solve this problem by diving the data into smaller groups typically in a scale from 10 to 100 items per page. Most of APIs that support pagination usually include this parameter in the request URL, so with this practice you specify which page of data you want to obtain, and those parameters include these specifications like page number and the number of items per page.

   - Example
     - Moving on to the different Pagination Methods, exist different types of it, with different uses,

- Offset Pagination
    - This type uses the `limit` and `offset` commands already present in the SQL library as query parameters.
    - To implement offset pagination, you can add two important query parameters like “limit” and “offset”.
        - “limit” basically specifies the number of items to return per page.
        - “offset” specifies the starting index of the items to return.
    - example of this type can be something like this.

```
@app. route('/products')
def get_products ():
    limit = request.args.get ('limit', 20, type=int)
    offset = request.args.get ('offset', 0, type=int)
    products = get_all_products ()
    return jsonify(products[offset: offset+limit])
```

    With this example, the ‘get_products ()’ function give all the product, from 0 to 20 items, in the Sam page.

    - Pros
        - You can jump to any page immediately, like skipping others results and only take care of the ones you want, this is not possible I the cursor-bases pagination.
    - Const
        - This kind of pagination does not scale at a database level, for example you want to skip 400,000 records and take the first 10 of it, so the database has to transverse all the 400,000 records before returning the 10, that you ask for, so this meaning has worst performance.
- Keyset Pagination
    - In this approach, a key or delimiter is selected by which data can sorted. So, this uses a unique identifier or key to paging through result. The API client used the last key of the previous page as the starting point for the next one and so on. Another point of this type of pagination can be more efficient that offset pagination when you are dealing with large datasets.
    - An Example of this can be:
        - The client request most recent item with `GET /items?limit=20`
        - Si until the next page were click (first page to second page), the query finds the minimum created date of 2023-02-23T00:00 with

this you can obtain the first 20 results, and this is used to create a query `limit` filter for the next page.

`GET /items?limit=20&created:lte:2019-01-20T00:00:00`
After all this will paginate until the last page is reached

- Cursor Pagination
  - Cursor-based pagination includes a cursor along with the requested page, which indicates the position of the next set of results, with this way the API can precisely manage the pagination logic.
  - Pros
    - Like we said previously the offset kind cannot scale, this type can scale with the records.
  - Const
    - You cannot jump to a specific page using only a cursor, basically means that you cannot guest which cursor presents the star of page 400 without first representing pages from 1 – 399.

3. **Describe the concept of callback function. 15 points.**

- Definition
  - A callback function is a function that passed an argument to another function is executed after that function has completed its task. The main purpose of used callback functions is to allow a program to respond to a different event or perform task in the background without *blocking the main thread of execution*. In other word it works like a function that is "*called at the back*" of the function it's passed into, or you can understand it better like "call after" function, because you going to used later eventually.
- Example

```
function greet(name, callback) {
  console.log("Hello, sir " + name + "!");
  callback();
}

function sayGoodbye() {
  console.log("Goodbye!");
}

greet("David", sayGoodbye);
```

    - As you can see, we have 2 functions 'greet' and 'sayGoodbye' , the 'greet' function takes two argument names, and callback that it's a function. How will use this functions, well something like this, we are trying to print your name and Goodbye and the same time and with only a single function call, and how we are doing that, well with this -> greet("Alice", sayGoodbye), because when we add name, and call our other function called sayGoodbye, this is possible because we declare our second argument like callback(), with this the computer interpret and know the used that going to do. After all this the printout o log, will be something like this.

```
Hello, sir David!
Goodbye!
```

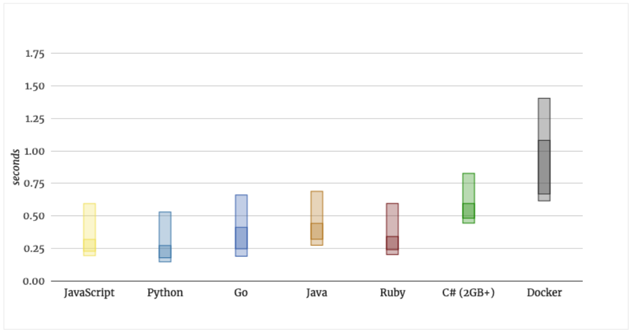
- Pros
    - Callback functions give you the power to run tasks in the background without blocking the other ones, or the main one.
    - Callbacks are widely used so most of the people are already familiar with them, then it's used in the tech industry.
- Cons
    - You need to be conscience of all the callbacks you used before, because can form something called "callback hell" or "pyramid of doom", when you have multiple nested callbacks, so making the code difficult to read and understand.
- Conclusion
    - In conclusion, a callback function is a power tool, that helps us to perform multiple operations at the same time, it needs to be used with care to not fail into the "callback hell", and the option to create more structure code to solve complex operations with this tool.

4. **Describe the concept of cold start in AWS Lambda. 15 points.**
    - AWS Lamba
        - AWS Lamba is a service that allows you to run without the need to provision manage server. It operates on a relievable compute infrastructure that provides computer resources.
    - Definition
        - This concept of cold star happens when the first request comes in after deployment, or when a function is invoked, or after initial invocation, it can happen with different situations, basically when this happens the

Lamba service needs to create a new execution environment for the function.

Another important info is cold start happen 5 to 7 minutes after previous request.

- Ways cold star can happen.
  - A function is invoked for the first time.
    - With this action AWS needs to allocate the resources needed to run the function that you want to, such can be CPU, memory, and network connections, all this process can take time to occur, fundamentally all the environments need to be set up before function can run.
  - After initial invocation
    - This happen when the function remains idle for a certain period, so basically happened that AWS dislocated the resources, and when you want to star this again, it's going to lead to another cold start.
- How slow are cold stars?



  - JavaScript, Python, Go, Java, and Ruby are all similar they usually finish within 400 milliseconds and almost always under 700 milliseconds.
  - C# is a bit of an underdog here. The chart shows stats for instances with 2+ GB of allocated RAM, which are faster than smaller ones. Cold starts for this instance size take between 0.4 and 0.9 seconds. Lambda functions packaged as Docker images are even slower.
- How can prevent a cold start?
  - Provisioned Concurrency: This is a feature that allows you to pre-warn a specific number of executions environments before the function is invoked, so with this you can reduce the impact of cold starts.

    - Reduce function package size: The size of you function package can also impact during the cold start, so its important to trying to reduce the size of it.
    - Increase function frequency: Cold stars typically occur after a period of inactivity. So, this is possible by setting up automated period invocations.

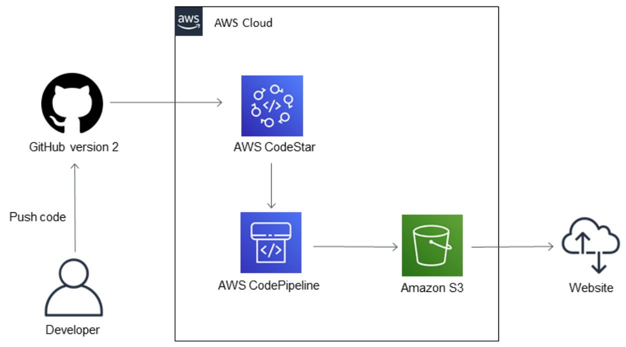
5. Describe each HTTP methods. 15 points.

- Definition
    - HTTP (Hypertext Transfer Protocol) basically is the foundation of data communication on the WWW (World Wide Web) and it is a client-server protocol, which means are initiated by the recipient usually the Web browser, and it was design for communication between our web browser and web servers, so a clearly example of this, is when a client opened a connection to make a request, the waiting until receives a response. It's important to clarify that the server does not keep any data between two requests.
- HTTP methods
    - GET Method
        - Basically, if you want to retrieve data form a resource website, like you do with servers API, we do a GET request, that means a GET request to the server, so inside that you need to specify all the specifications you want.
    - POS Method
        - This method creates a new resource on the backed our server, this body carries the data we want to the server, in other words to summit form data or to upload a file to the server.
    - PUT Method
        - With this method you can update and existing resource by sending the updated data of the request of the body to the server
    - PATCH Method
        - This method it's not commonly used, because is like the PUT method, but the difference it's that update the provide fields

of the format like JSON or XML, to say an example like customer identify, with PUT method you will update the customer on the resource entirely and not partially.

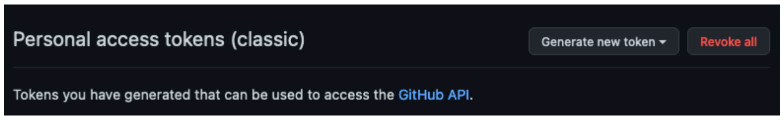
- Delete Method
    - The Delete method basically delete the resource from the server entirely, and it doesn't matter the number of call the result is going to be the same. When the process is done, the server can choose to mark the resource as deleted or to move it to a trash folder, that means permanently deleting it.
- HEAD Method
    - This one is particularly like GET method; the main difference is this one does have any response body. This one it used more in metadata about the resource because you can obtain size, type, data all this without downloading its entire content.
- OPTIONS Method
    - This method is used to get to get information about the possible communication options for the given URL in the server. This return with a list of available options that the client can use to refine its request.
- TRACE Method
    - Is for diagnosis purposes, this echo back the received request message to client, with this the client can see things intermediaries (proxies, gateways, etc.), is used for debugging and diagnostics options. This method could be dangerous because if you don't have security, it could reveal credentials.
- CONNECT Method
- This method enables end-to-end connections between a client and a server over HTTP, creating a two-way bridge for secure communication through an HTTP proxy server. One practical example of this method is its use in safely transferring large files between client and server.

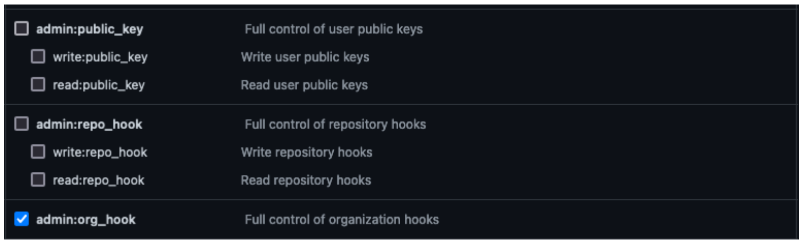
6. **Describe how you can automate a deployment of a static website to S3. 15 points**



- A automate deployment will like something like the image above.
- To create this automation of a deployment we need to do creating in step first.
- First able, Create an S3 bucket, if we haven't yet, and of course ensure that our bucker properties are configured to allow static website hosting. And you must already create a GitHub account and GitHub repository.
- Of course, have a static repository to be saved.
- Create a build pipeline using a Continuous Integration/Continuous Deployment (CI/CD) this mean tool such a Jenkins, Travis CI, and Code Pipeline, so us we going to center our focus with CodePipeline.
- Now, that we going to start building our JSON script its important to understand that our pipeline is going to start once a new commit is made on the configured GitHub repository or the branch that we going to use.
- Create a pipeline, to get to that we need to create a JSON file.

```
{
    "pipeline": {
        "name": "my-pipeline",
        "roleArn":
"arn:aws:iam::292274580527:user/DAVIDE.ROLDAN@CETYS.E
DU.MX",
        "artifactStore": {
            "type": "S3",
            "location": "roldan.cetystijuana.com"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "SourceAction",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "ThirdParty",
                            "provider": "GitHub",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "Owner": "Magnus1515",
                            "Repo":
"Roldan_Cloud_Website",
                            "Branch": "main",
                            "OAuthToken":
"Personal_access_token"
                        },
                        "outputArtifacts": [
                            {
                                "name":
"SourceOutput"
                            }
                        ],
                        "inputArtifacts": []
                    }
                ]
            },
            {
                "name": "Deploy",
                "actions": [
                    {
                        "name": "DeployAction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName":
"roldan.cetystijuana.com",
                            "Extract": "true"
                        },
                        "inputArtifacts": [
                            {
                                "name": "BuildOutput"
                            }
                        ]
                    }
                ]
            }
        ],
    }
}
```

- The important concepts of this JSON will be the next ones:
  - pipeline
  - "roleArn": "arn:aws:iam::80398EXAMPLE::role/AWS-Code Pipeline-Service" -> Means the Amazon Resource Name to identify he roles permissions to AWS, but we don't have permissions for that.
  - Location -> where it's your bucket.
- Stages
  - "name": "Source" -> Indicates the name of the stage.
  - "category": "Source" -> This category defines what kind of action can be taken in the stage, it can be like (source, build, test, deploy, invoke, approval). Us we want to use Source.
  - "runOrder":1 -> The order in which actions are run.
  - "OAuthToken": "Personal_access_token" -> you can get your personal access token in GitHub settings/Developer settings/Personal access tokens/Tokens(classic).
    You need to create a new token in the button.
    
    Personal access tokens (classic)

    Tokens you have generated that can be used to access the GitHub API.
    In all the checkbox of the token you need to select the most important one.
    
    ☐ admin:public_key — Full control of user public keys
    ☐ write:public_key — Write user public keys
    ☐ read:public_key — Read user public keys
    ☐ admin:repo_hook — Full control of repository hooks
    ☐ write:repo_hook — Write repository hooks
    ☐ read:repo_hook — Read repository hooks
    ✓ admin:org_hook — Full control of organization hooks
- Deploy
  - name": "Deploy" -> Indicates the name of the stage.
    - "name": "DeployAction" -> Now we going to do a DeployAction.
  - "name": "DeployAction",

        - "category": "Deploy", -> Now we going to use a Deploy related with our S3.
        - "owner": "AWS", -> The owner of the tool.
        - "provider": "S3", -> our provider this time is S3.
        - "version": "1" -> The version that its running.
        - Configuration
            - "BucketName": "roldan.cetystijuana.com" -> the name of the bucket.
            -

    - Once we already create our JSON file and saved it for example like pipeline.json
    - Now we need to run the next command adding our JSON file.

```
aws codepipeline create-pipeline --cli-input-json file://pipeline.json
```

- Just to finish now we need to make our pipeline to start working, for that, we going to use the next command.

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

    - Now all the commits that we do in our GitHub repository will also be done in our bucket too, so it will be connected, this way we will have an easier way to have everything updated.

7. **Read the Real-world Engineering Challenges #8: Breaking up a Monolith article and write a summary and opinions about it. 10 points.**

Additionality this article describes complete the process of how the migration from one language to another was, it was from Python to Go, the describe how was the process the technique they used, why it was necessary that migration, why they decide about that language, if was wort it the transition, and the lessons learned from this migration. Adding to this resume it was really interesting the comments of personas who worked antireality in this process, Brian and Kevin add their thoughts about this journey.

- Important ideas about this article
    - Before you consider a language, you need to test it against your other languages that you think will be a good idea.

- Under this migration you need to consider the libraries that you used before and the compatibility that they have with you knew language that you consider to.
- How precisely was consider the time for the deadlines (considering the stage of before starting, MVE and endgame)and how they had achieved them. This could sound a little bit logical for us, but this features its kind a hard to guess.
- The organization they did to rewrite all this code, basically they avoid a "big band" migration. Instead. They used "field by field.
- How they prepare to use Go because all most no one have knowledge in this new language.
- How was the process of the migration to rewrite in the same process if the website its online, this process is complicated because you need to have a great control of the user's traffic.
- If was important to know that other companies don't use agile, I know that exist other methods, they use a simple one was a fixed-scope, fixed timeline project, where they had a massive burndown chart that always gave them a good understating of how where they are going.
- Sometime the hard deadlines can be motivational, but do not overdo it because it does not create a good working environment.

At the end of the process, you need to analyze if all the changes that you did through the journey were worth it, here emphasize that everything it's going to be that you cannot control or anticipated, maybe you can reduce them, but every time is going to appear a new one challenge and you need to face them in the best way.