Laboratory Report

Version control and continuous deployment

20/10/23

Engineering in Information Securtity and Networking

Cloud architecture

Prof. Mtro. Rodolfo Luthe Ríos

Edgar Guzman Claustro

Si727576@iteso.mx

ITESO

Universidad Jesuita
de Guadalajara

Abstract

During the development of this laboratory practice, four functionalities of the Rekognition API will be demonstrated. Starting with the face comparison, where the API is going to detect similarity between to faces images. The second functionality is the celebrity recognition, the API's answer is the name and the similarity percentage of similarity with the famous person in the photo. The third functionality is the text detection inside the image, where the API is returning the text. Finally, the face detection, where the API is capable to detect faces and return their emotions and more characteristics.

The recognition API works with AI and the simple part of all is that the user does not need to be a programmer or an IA data engineer. Just to make the simple call of the API with the correct parameters.

Conceptual framework

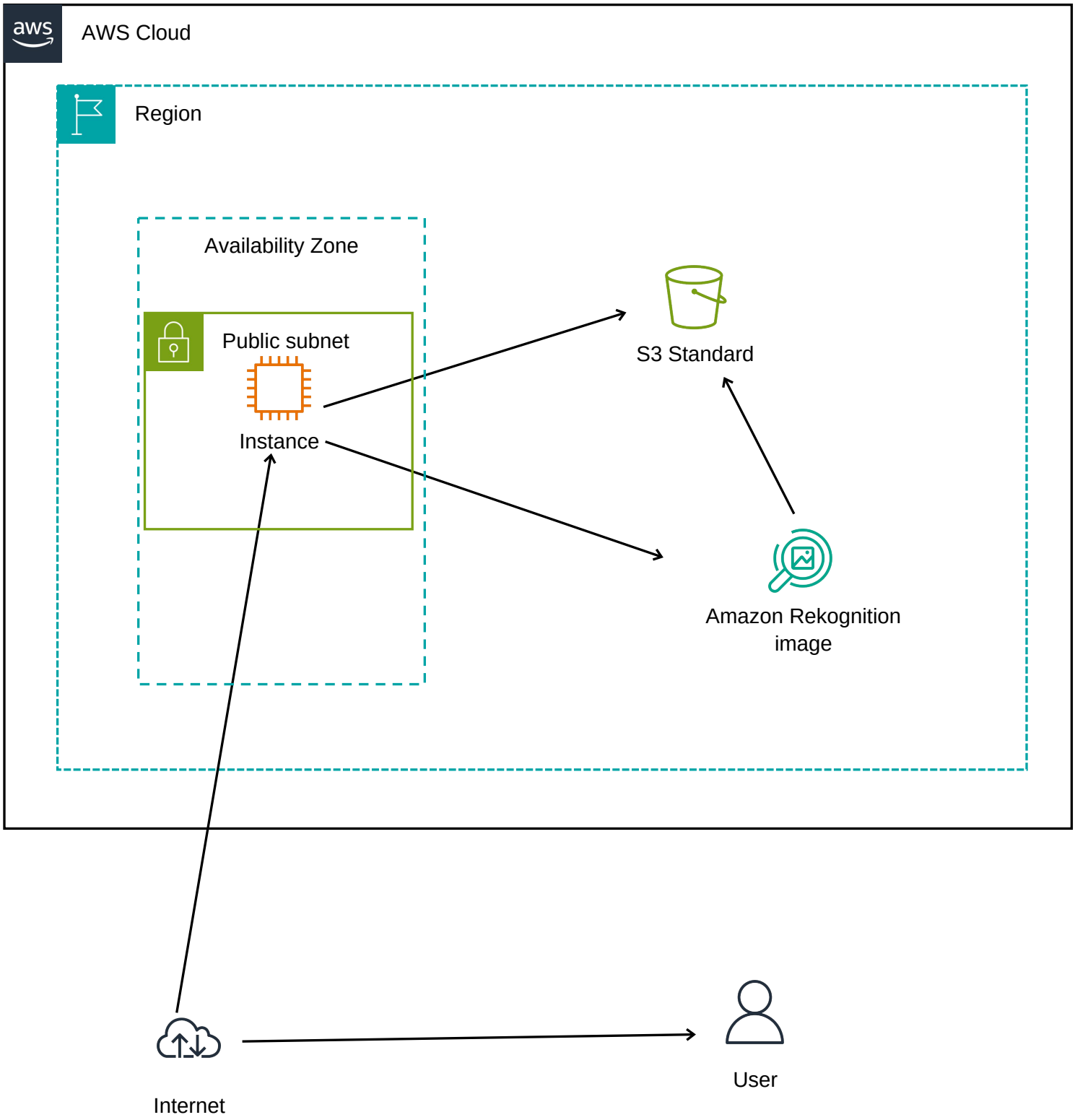
Cloud infraestructure: Cloud Infrastructure is the collection of hardware and software elements such as computing power, networking, storage, and virtualization resources needed to enable cloud computing. Cloud infrastructure types usually also include a user interface (UI) for managing these virtual resources. [1]

API: APIs are mechanisms that allow two software components to communicate with each other using a set of definitions and protocols. For example, the meteorology institute's software system contains daily weather data. The weather app on your phone “talks” to this system through APIs and shows you daily weather updates on your phone.[2]

Rekognition: Amazon Rekognition offers pre-trained and customizable computer vision (CV) capabilities to extract information from images and videos.[3]

S3: Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. [4]

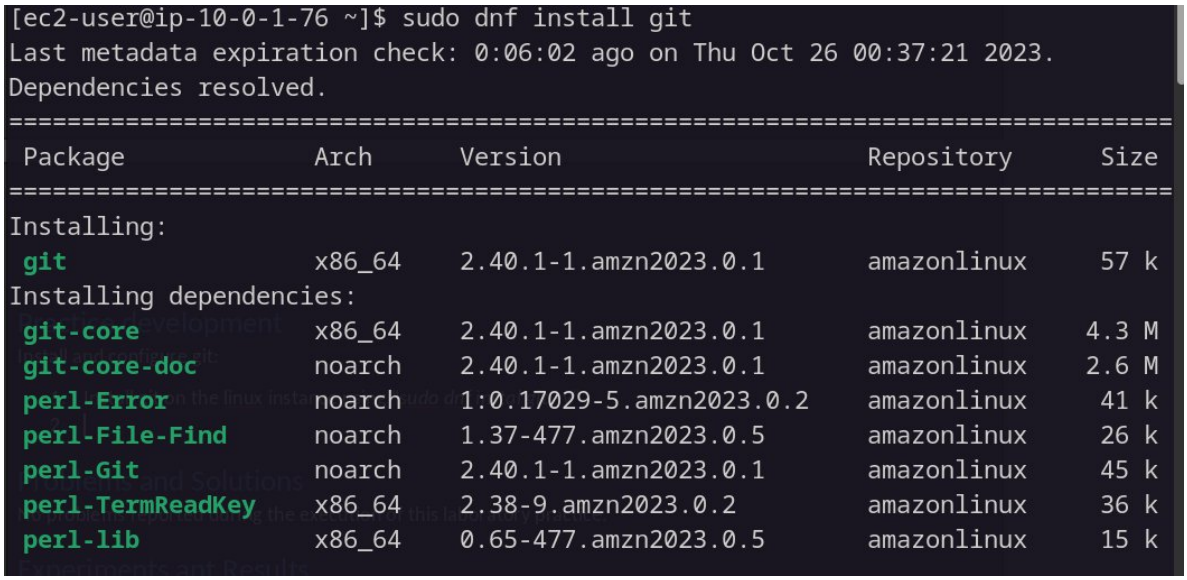
Diagram


AWS Cloud
Region
Availability Zone
Public subnet
Instance
S3 Standard
Amazon Rekognition image
Internet
User

Practice development

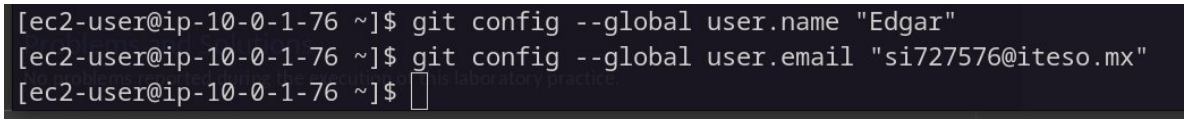
Install and configure git:

1. Install git on the linux instance using "*sudo dnf instal git –y*"



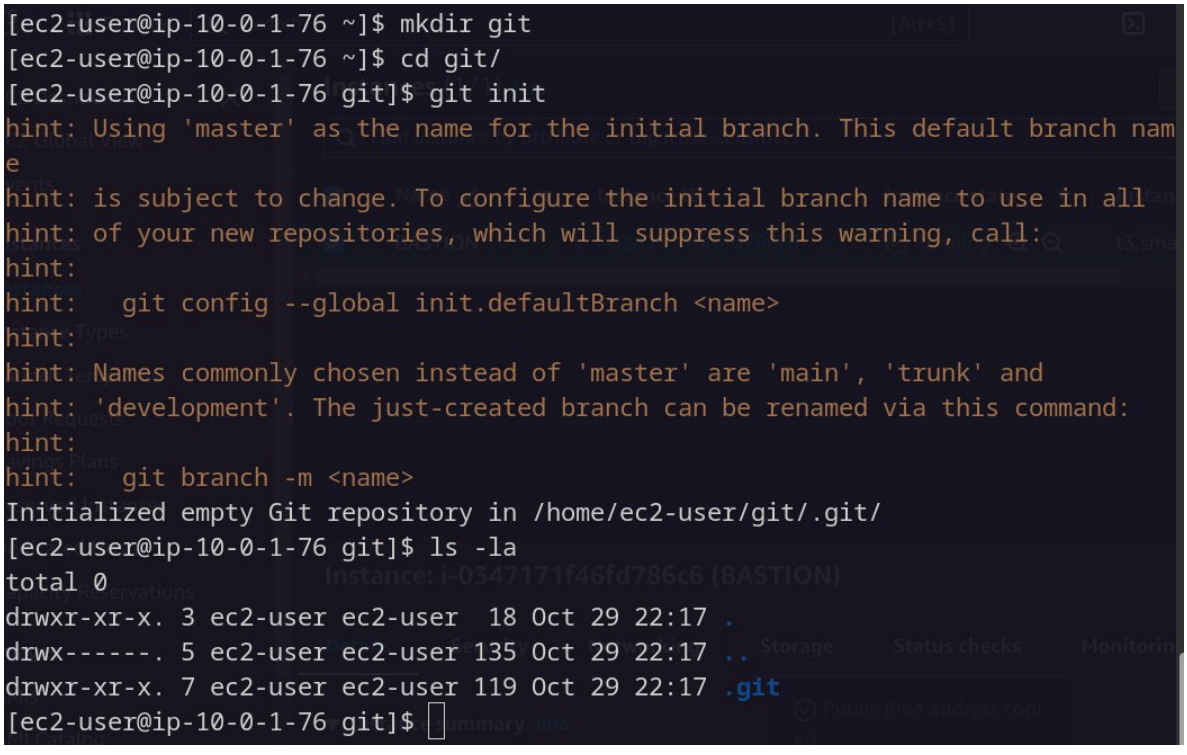
```
[ec2-user@ip-10-0-1-76 ~]$ sudo dnf install git
Last metadata expiration check: 0:06:02 ago on Thu Oct 26 00:37:21 2023.
Dependencies resolved.
================================================================================
 Package             Arch      Version                     Repository      Size
================================================================================
Installing:
 git                 x86_64    2.40.1-1.amzn2023.0.1       amazonlinux     57 k
Installing dependencies:
 git-core            x86_64    2.40.1-1.amzn2023.0.1       amazonlinux    4.3 M
 git-core-doc        noarch    2.40.1-1.amzn2023.0.1       amazonlinux    2.6 M
 perl-Error          noarch    1:0.17029-5.amzn2023.0.2    amazonlinux     41 k
 perl-File-Find      noarch    1.37-477.amzn2023.0.5       amazonlinux     26 k
 perl-Git            noarch    2.40.1-1.amzn2023.0.1       amazonlinux     45 k
 perl-TermReadKey    x86_64    2.38-9.amzn2023.0.2         amazonlinux     36 k
 perl-lib            x86_64    0.65-477.amzn2023.0.5       amazonlinux     15 k
```

2. Configure the user "*git config --global user.name "Edgar""* and "*git config --global user.email "si727576@iteso.mx*""



```
[ec2-user@ip-10-0-1-76 ~]$ git config --global user.name "Edgar"
[ec2-user@ip-10-0-1-76 ~]$ git config --global user.email "si727576@iteso.mx"
[ec2-user@ip-10-0-1-76 ~]$
```

3. Setup the repository, create a folder callet git and run "*git init*".



```
[ec2-user@ip-10-0-1-76 ~]$ mkdir git
[ec2-user@ip-10-0-1-76 ~]$ cd git/
[ec2-user@ip-10-0-1-76 git]$ git init
hint: Using 'master' as the name for the initial branch. This default branch nam
e
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ec2-user/git/.git/
[ec2-user@ip-10-0-1-76 git]$ ls -la
total 0
drwxr-xr-x. 3 ec2-user ec2-user  18 Oct 29 22:17 .
drwx------. 5 ec2-user ec2-user 135 Oct 29 22:17 ..
drwxr-xr-x. 7 ec2-user ec2-user 119 Oct 29 22:17 .git
[ec2-user@ip-10-0-1-76 git]$
```

4. Verion control. Create a document named "vresions.txt" and write "version 1". Run "*git add .*" and "*git commit*".

```
[ec2-user@ip-10-0-1-76 git]$ nano Versions.txt
[ec2-user@ip-10-0-1-76 git]$ git add .
[ec2-user@ip-10-0-1-76 git]$ git commit
[master (root-commit) 6be40a4] Version 1 sssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssss
 1 file changed, 1 insertion(+)
 create mode 100644 Versions.txt
[ec2-user@ip-10-0-1-76 git]$
```

5. Modify again the file and write "*Version 2*" run "*git add .*" "git commit –m"

```
[ec2-user@ip-10-0-1-76 ~]$ cd git/
[ec2-user@ip-10-0-1-76 git]$ ls
Versions.txt
[ec2-user@ip-10-0-1-76 git]$ nano Versions.txt
[ec2-user@ip-10-0-1-76 git]$ git add .
[ec2-user@ip-10-0-1-76 git]$ git commit -m "version 2"
[master 094d7c3] version 2
 1 file changed, 1 insertion(+), 1 deletion(-)
[ec2-user@ip-10-0-1-76 git]$
```

6. Do the same thing with version 3

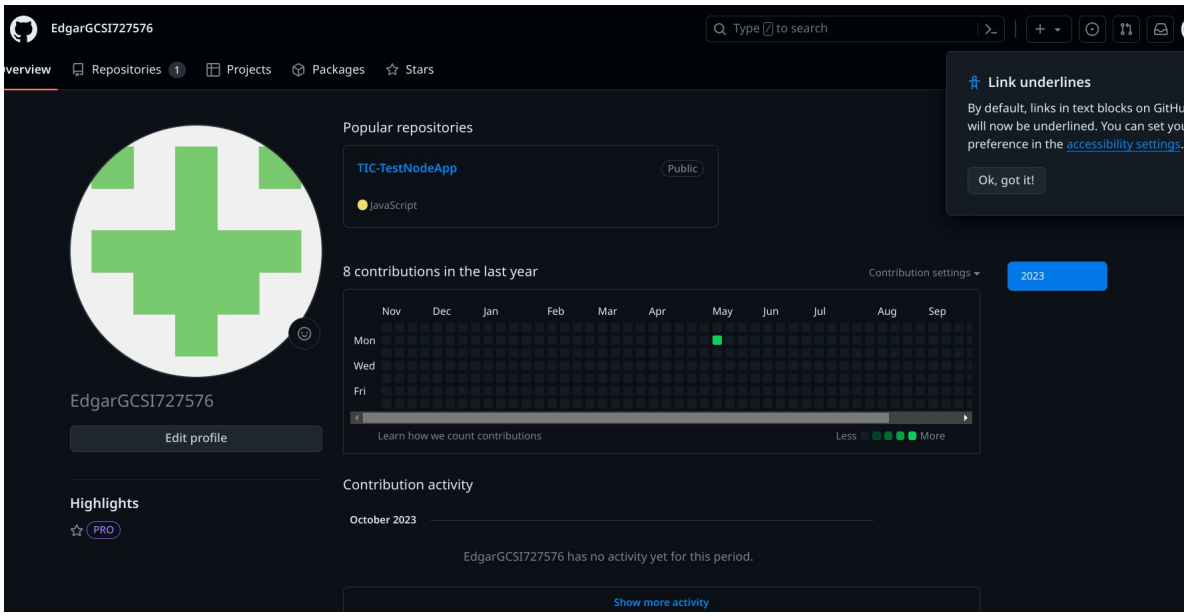
```
[ec2-user@ip-10-0-1-76 git]$ nano Versions.txt
[ec2-user@ip-10-0-1-76 git]$ git add .
[ec2-user@ip-10-0-1-76 git]$ git commit -m "versions 3"
[master b2e4753] versions 3
 1 file changed, 1 insertion(+), 1 deletion(-)
[ec2-user@ip-10-0-1-76 git]$
```

7. Check versions history running "*git log*".

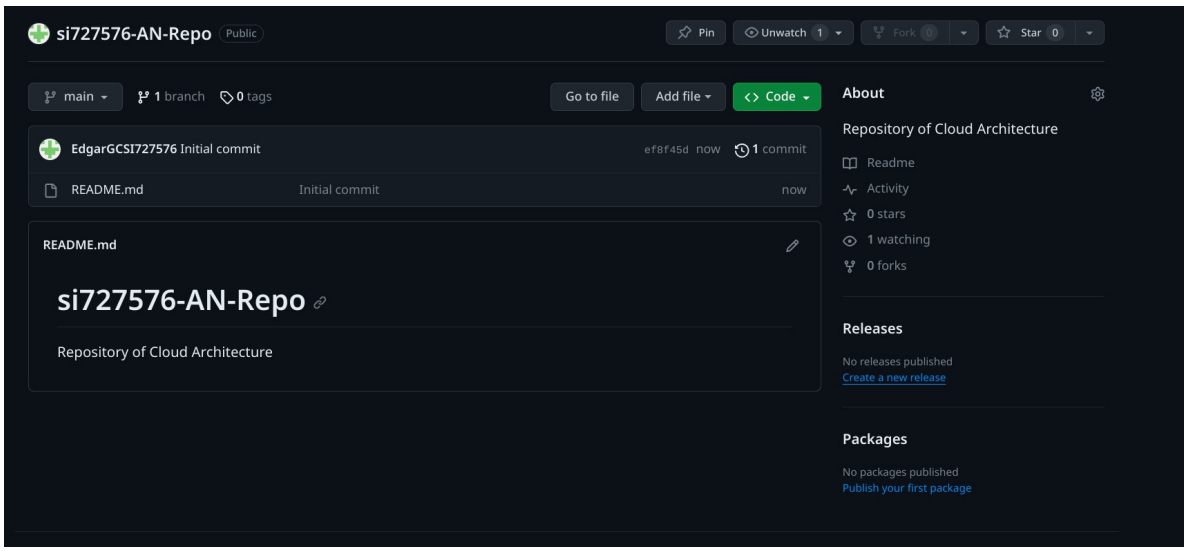
```
[ec2-user@ip-10-0-1-76 git]$ git log
commit b2e47535b17ee668e86575732e17e341b5e348b1 (HEAD -> master)
Author: Edgar <si727576@iteso.mx>
Date:   Sun Oct 29 23:08:47 2023 +0000

    versions 3

commit 094d7c3ecc522c32184fe7236a34b890f3ff8c31
Author: Edgar <si727576@iteso.mx>
Date:   Sun Oct 29 23:07:09 2023 +0000

    version 2

commit 6be40a444706d685a46b61d0f282bb71414609c2
Author: Edgar <si727576@iteso.mx>
Date:   Sun Oct 29 22:19:43 2023 +0000

    Version 1
```

Create GitHub repository.

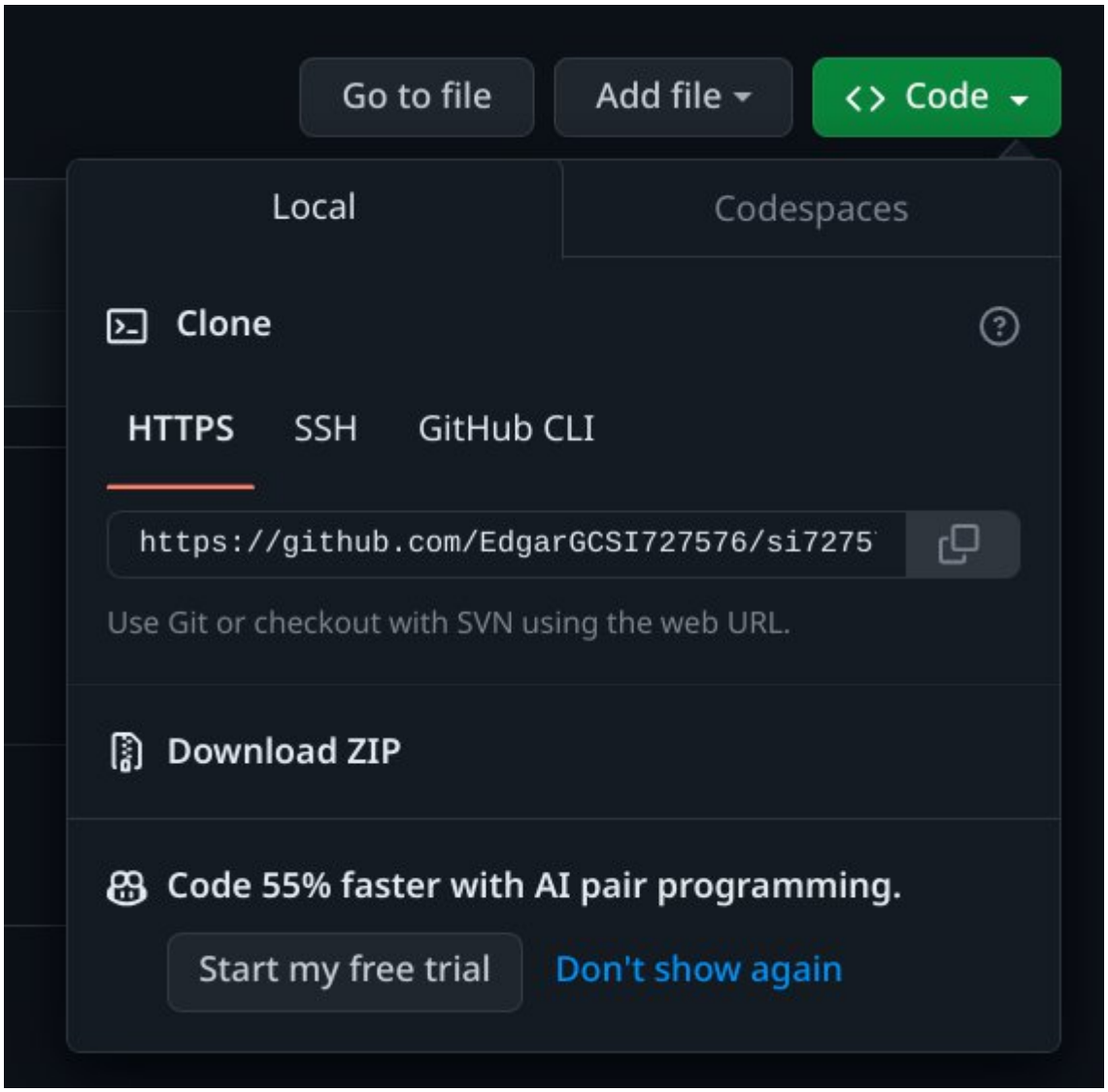
1. Create an account with the ITESO email on GitHub.



2. Create a repository.



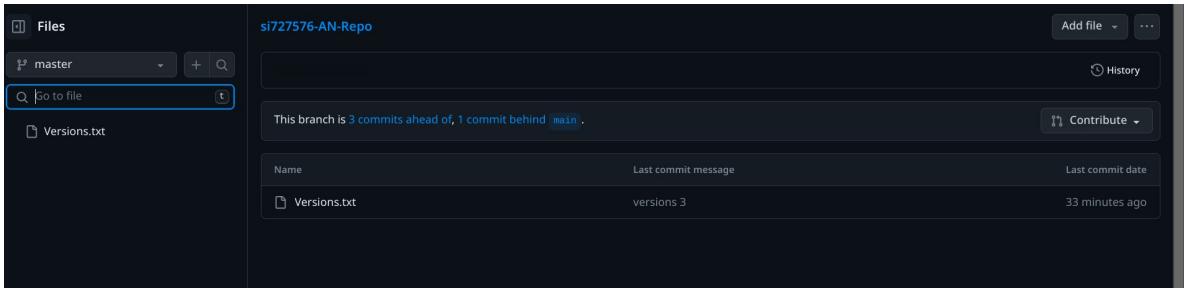
3. Check the URL of the repository.



4. Add the GitHub repository to the local repository, from the git folder: “*git remote add Hub <URL>” “git push Hub master”.*

```
[ec2-user@ip-10-0-1-76 git]$ git push Hub master
Username for 'https://github.com': EDGARGCSI727576
Password for 'https://EDGARGCSI727576@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (9/9), 654 bytes | 654.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/EdgarGCSI727576/si727576-AN-Repo/pull/new/master
remote:
To https://github.com/EdgarGCSI727576/si727576-AN-Repo.git
 * [new branch]      master -> master
[ec2-user@ip-10-0-1-76 git]$
```

5. Refresh the page and see the changes



Control versions of a document

1. Create the report document, and add it for tracking.
2. Make and add commit for each section of the document.
3.

Problems and Solutions

No problems reported during the execution of this laboratory practice.

Experiments ant Results.

The result are very interesting. The result of the first part of the practice was an API that is capable to compare faces. In the given example the API was provided with two images of the same person, an actual photo and another one when the person was younger. The respond of the API was over 90% of similarity between the images.

Furthermore, the second part of the laboratory was kind of more interesting. Here three different types of the functionality of the API were tested. Beginning with the “celebrity recognition” a photo of Kanye West was provided, and the respond was successfully correct. Then, the second image was given to the text analyzer functionality, where the respond of the API will be the text of the image. Here the results where a little bit wrong, the answer was not complete. Finally, a photo with different face expression was provided and the answer of the API were the number of faces and the expression.