Computer Systems Engineering

Cloud Computing

Course Project

Marco Ricardo Cordero Hernández – IS727272

Tlaquepaque, Jal., November 29, 2023

Index

Requirements 1
Theoretical Framework 2
Architecture 4
    Architectural diagram 4
    Architecture rundown 4
Development 6
    Initial approach 6
    Setup 7
    Web server 7
    Services implementation 10
Problems and solutions 16
Experiments and results 18
Cost analysis 19
Conclusions 20
References 21

Requirements

The goal of the current development is to demonstrate an implementation of a URL shortener (formally introduced later on) implemented inside a cloud provider with remotely located resources. In simple terms, the goal of said tool is to act as a black box which takes a long string of characters representing a remote resource location over the Internet, and as the result a short form of said string will be returned. For example, a long Google search term URL can be reduced to even less than twenty characters, this of course varying on each shortening technique.

There are several motivations and personal objectives found within this development, however, those that stand out from the rest have something in common: the desire of taking an apparently intrinsic technology, dissecting it for better understanding, and the deploying an own proposal. This of course can transgress the mere academic purpose of the current project, going as far as to upgrading several basic components of this proposal in order to establish a fully supported solution than can successfully compete with real world, this being evidently achieved by its commercialization.

Before digging deeper into the impactful possibilities of this work, the academic purposes can't be overlooked, as the contents present reflect the progress made so far and the prowess acquired for cloud handling and deployment. It is important to remember how much of an agent of change have remotely architectures played in major businesses, overtaking the current development world by providing resources on-demand and on the fly, a pair of topics that have been thoroughly discussed previously. Even when multiple enterprises nowadays still might opt for on-premises infrastructure over the cloud (yet having the possibility of a hybrid approach), it can't be denied that remote computing and several other services hosted in a certain kind of cloud are present and are widely used at the moment of this writing; of course it's still important to know about concepts and practices of physical, tangible infrastructure, but now both might as well be as relevant in the technology compendium of the average IT professional.

Going solo into developments such as the one that is about to be introduced might not seem that overwhelming, however, one relevant point of cloud infrastructure is that, at its most simplistic core, using, integrating, or altogether migrating to remote hosted services it's basically getting rid of the physical component of a development, but that doesn't automatically solves the need for special and dedicated distributed teams to focus on specific business needs. In fact, cloud usage should facilitate remote collaboration, but that's another discussion aimed for an alternative space.

Theoretical Framework

Back in the decade of the 1960's, the mere concept of wireless connectivity probably would've been mocked upon and perhaps seen as the idea of someone far too ambitious, maybe even as to consider them deranged. Back then, the ARPANET [1] was one of the primary bets the government had put the contributors' money on, and surely it will come to great results when public networking began its roots, penetrating into scientific and academic fields, and eventually, into the public domain. Before the common internet user would be able to sign a contract with an ISP, the ARPANET, such as other emergent networks were already having troubles with domain handling to resolve local and external domains, this primarily being by the switch of the NCP network protocol to the now widely used TCP/IP [2].

In 1983, the Domain Name System (DNS) was proposed, and so, network resources would be arranged hierarchically, however, standardization was yet to come. Almost 10 years later, in 1992, Tim Berners-Lee along with his team created three fundamental things that would become the pillars of the modern web: the HyperText Markup Language (HTML) to display pages, the HyperText Transfer Protocol (HTTP) to send and receive resources over the internet, and the ***Uniform Resource Locator*** (URL). Lee, while working at the European Council for Nuclear Research (CERN), joined these technologies to create the World Wide Web, the known system to access interconnected public web pages distributed across the Internet (Web ≠ Internet) [3].

All the components that conform the web are equally important, however, the point of focus and the interest of the current work relays on URL's. As their name suggest, these elements are addresses corresponding to web resources and they're often found in the form of hyperlinks, a format which can correlate any text string with a URL [4]. The inspiration for the creation of the latter came from another problem that URL's created: extensiveness. As the domain managing problems became of less importance, flooding the emergent web with resource was something foreseeable, however, long URL formats perhaps were not.

Moving into modern times, the web has become plagued with several high throughput means of information consuming in fast-paced work environments, and the problem of long formats persists and linger more than before, and that is without taking into account extended URL's parts such as ports, queries and fragments [5]. In 2002, Kevin Gilbertson faced a similar issue when he wanted to share his pictures over web pages, but didn't liked the idea of having long strings for something that should be straightforward; and so, TinyURL was born, the first known *URL Shortener* [6].

The definition of a URL Shortener couldn't be simpler, because as the name indicates, this type of tool takes a variable length URL, trims it, and a new shorter URL it's returned. Simple, at least in theory. But, after all these concepts have all been formally introduced, perhaps the next suitable question would by: why would these services exist when hyperlinks already tackle this problem in a highly flexible form? The thing is, not all data representations allows for hyperlinks.

Back when the social platform *X* was known as Twitter, the initial characters length limit was set at 140 per post, later on it doubled to 280, but still, the rants known as threads still require several posts to cover all the details of the most recent gossip; this fact, either by being purposely inducted to generate more interactions, or by truly abiding by the problematic scenario essence, makes the scarcity of available digital resources something to take into account. Take a look at the references at the end of this project, how many characters would the longest of these URL take from one post?

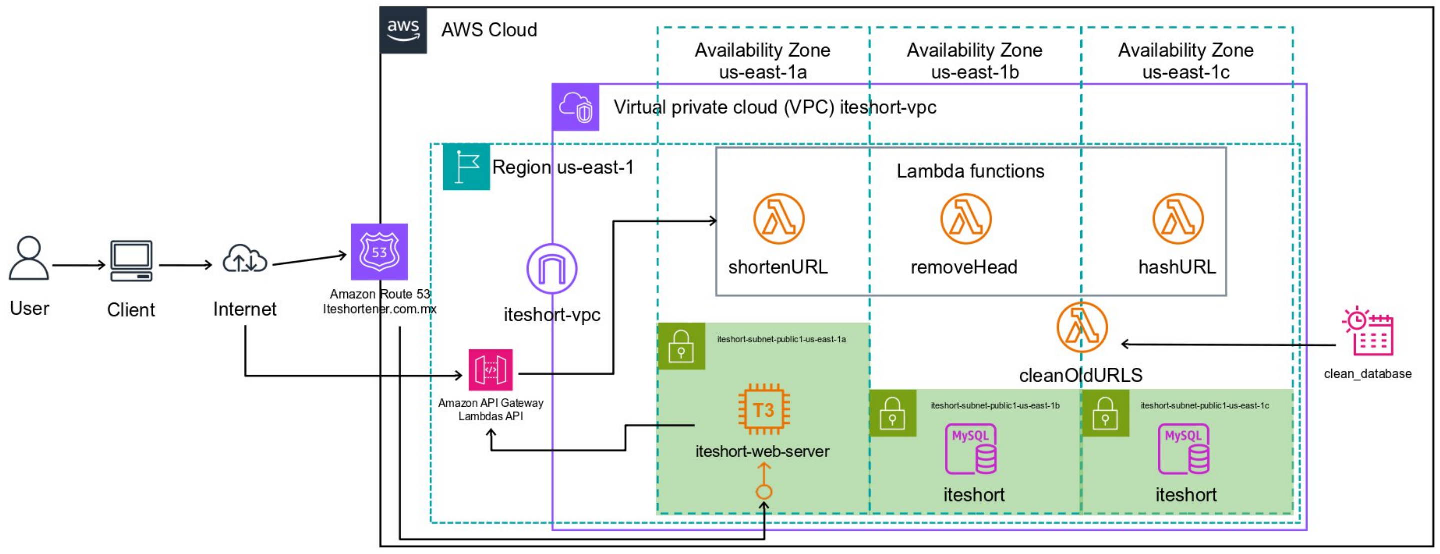
Yes, these might be trivial, but brevity is something that extends far outside social interactions, for example, modern merchandising, in which ad campaigns are designed to be short and attract the sight of a vast audiences, either digitally or directly in person. The first case is covered, hyperlinks can be useful there, but, for a poster outside a store or a billboard, a way of compressing data and present in a quick way it's vital. Surely now there are other alternatives such as Quick Response (QR) codes, but even those have a limit of around 4000 characters [7]. The main problematic isn't the limit for that case, because, if a URL has more than that amount of characters in its contents, surely there's another kind of issue involved that might not concern shortening; the main problematic it's that QR's require a reader to interpret the embedded data, which might come as a slight inconvenience, as most of modern mobile devices' cameras have this function integrated natively. Taking again the premise of a billboard, the short attention span of modern people and the implications of driving responsibly would make the reading of a QR while on the road something troublesome, unless a passenger is traveling along.

Even in idealized scenarios, URL's are the prominent way of representing a resource address. Surely a photo of a QR can be snapped to be read later, but, can it be written down? Can it be inserted as a hyperlink? Can it be typed? Can it be voiced? Perhaps one day, but at the moment, it can't be done. Given all the context that serves a double purpose for inspiration, the current project has the main goal of replicating the service that URL shorteners offer by deploying a web application with its infrastructure supported by cloud technologies. This will demonstrate not only a great core services understanding level of cloud infrastructure and management, but also the sufficient expertise needed to replicate existing technologies to give them some amount of customization.

Architecture

The last thing to consider before demonstrating the development it's to know how it will be implemented. As in past works, the quintessential cloud provider of choice it's Amazon and it's Web Services (AWS) [8], this because of its vast catalog, availability features and fairly low cost.

Architectural diagram



Architecture rundown

– Route 53 [9]: This service *could* be used to bring a previously bought domain or to buy a new one, however, the means through which this implementation is made makes impossible for domain registration, as the account in use is federated. Route 53 is like any other domain registry service, a domain is bought and then is used to serve its intended purpose. A theoretical demonstration will be shown.

– API Gateway [10]: As the name suggest, this service will provide the API to interact with the visual application and its routes.

– Internet Gateway [11]: Although not a service, this component is as important; it will provide Lambda access for ease of handling.

– EC2 [12]: One of AWS core services, will be used to deploy and manage the web application.

– Lambda functions [13]: Following a somewhat serverless infrastructure, a couple implementations of this service will be use to handle the URL conversion and serving, and communication to the database.

- MySQL [14]: The simplest of the database that can be used for little cost and ease of access.
- EventBridge [15]: A cloud monitoring service that will aid in URL expiration.
- Elastic IP [16]: Using elastic ip's will be useful specially for the EC2 instances, as its contents are desired to be on the same location always.
- Network components and others: Also not services but VPC [17], regions and availability zones [18], and subnets [19] will significant roles inside the architecture.

Development

Initial approach

Before proceeding with the technical part of the project, some considerations need to be done, such as the scope of the objectives. URL shortening is easy to understand but hard to perfect, as several aspects of it have to be considered, which, for the current project, might produce an overkill. The current objective is to demonstrate a functional application with its infrastructure implemented on the cloud and nothing else. If cloud development was a priority, this would be a different story, but is not.

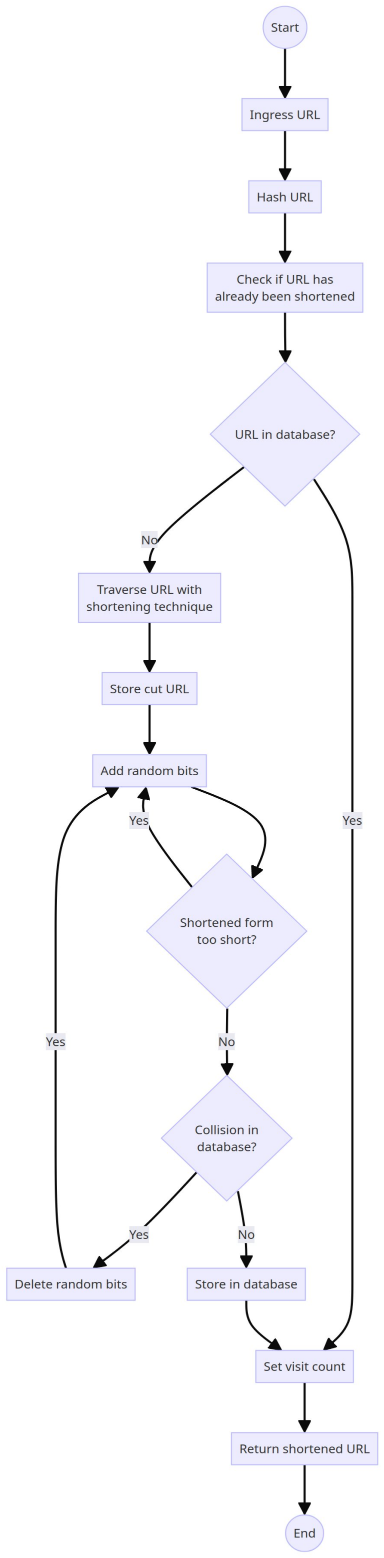
Based on the understanding set by the previously mentioned accord, the minimum requirements for the service to implement include (but are not limited to):

- URL formatting
- URL hashing
- Duplication avoidance
- Collision detection and handling
- Data preservation
- Data expiration

The stated services are capable of covering these requirements as they're listed and almost in a seamless way.

The diagram found at the left describes a simple workflow tackling some of the requirements mentioned. Technologies and services are omitted to obtain a better degree of abstraction, facilitating the implementation by removing restrictions and redundancy.

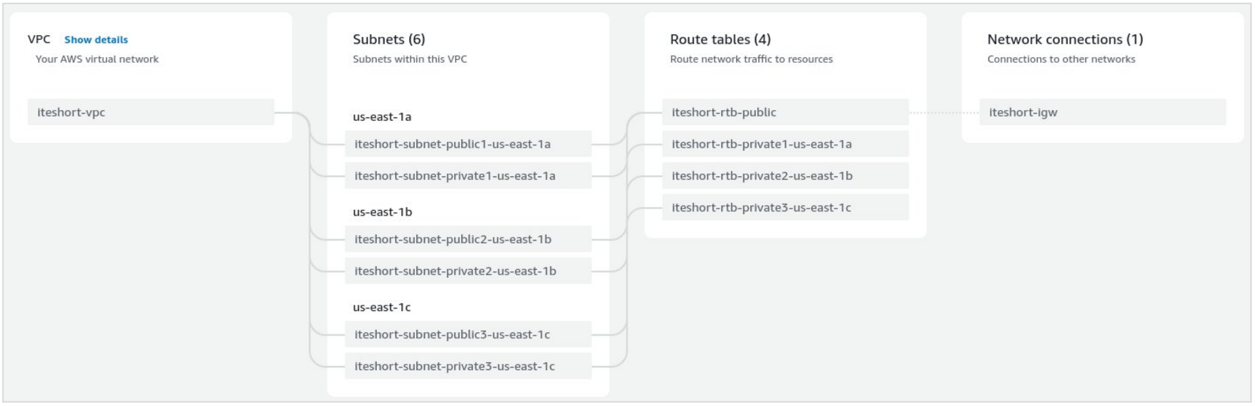
As already stated, the next parts will cover the sufficient conditions in order for the URL shortener to operate as expected, but the improvement areas are also acknowledge, such as better UI design, dashboards for SEO purposes, user handling for customized URL's, and many other features. Also, as already established, these concerns go far beyond the main goal, but they might open a discussion space to cover their topic contents later on the document.



Setup

Before going with formal service configurations, previous settings are required, mainly those of network. First, the VPC will be created through it's own section inside the AWS console; this will facilitate some settings, as other network resources can also be configured in the same utility:

- VPC and more
- Name: iteshort (auto-generate option enabled)
- IPv4 CIDR block: 10.0.0.0/16 (No IPv6)
- Tenancy: default
- Availability Zones: 3 (us-east-1a through 1c)
- Public and private subnets: 3
    - Public subnet 1a - 10.0.1.0/24
    - Public subnet 1b - 10.0.2.0/24
    - Public subnet 1c - 10.0.3.0/24
    - Private subnet 1a - 10.0.10.0/24
    - Private subnet 1b - 10.0.20.0/24
    - Private subnet 1c - 10.0.30.0/24
- No NAT gateways
- VPC endpoints: S3 Gateway
- Enable both DNS options


VPC Show details
Your AWS virtual network
iteshort-vpc
Subnets (6)
Subnets within this VPC
us-east-1a
iteshort-subnet-public1-us-east-1a
iteshort-subnet-private1-us-east-1a
us-east-1b
iteshort-subnet-public2-us-east-1b
iteshort-subnet-private2-us-east-1b
us-east-1c
iteshort-subnet-public3-us-east-1c
iteshort-subnet-private3-us-east-1c
Route tables (4)
Route network traffic to resources
iteshort-rtb-public
iteshort-rtb-private1-us-east-1a
iteshort-rtb-private2-us-east-1b
iteshort-rtb-private3-us-east-1c
Network connections (1)
Connections to other networks
iteshort-igw

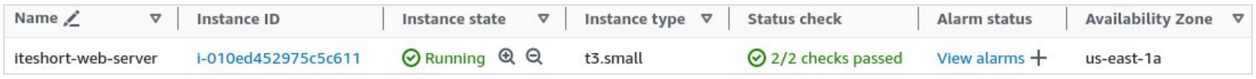
This automatically created network resources should suffice the current needs.

Web server

The next step will be setting up the EC2 with it's corresponding elastic IP. The instance features are the following:

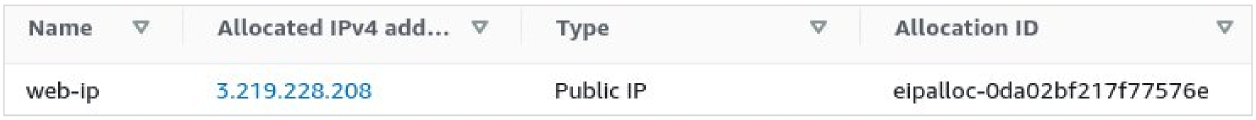
- Name: iteshort-web-server
- OS: Amazon Linux 2023 (or newest) AMI

- Instance type: t3.small
- Key pair
  - Create with name: iteshort-key
  - RSA with .pem format (for SSH connectivity)
- Network
  - VPC: iteshort-vpc
  - Subnet: iteshort-subnet-public1-us-est-1a
  - Auto-assign public IP: Disable (elastic IP will later be used)
  - Security group
    - Name: iteshort-server-sg
    - Set SSH and HTTP/s inbound rules from anywhere
- Storage: 8gb or minimum amount, gp2

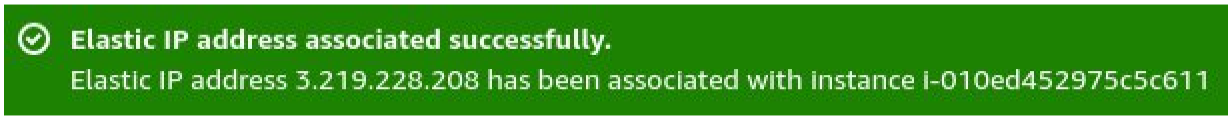


Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
iteshort-web-server	i-010ed452975c5c611	Running	t3.small	2/2 checks passed	View alarms	us-east-1a

After this, allocate a new elastic IP address with default options and then associate it with the EC2 instance.

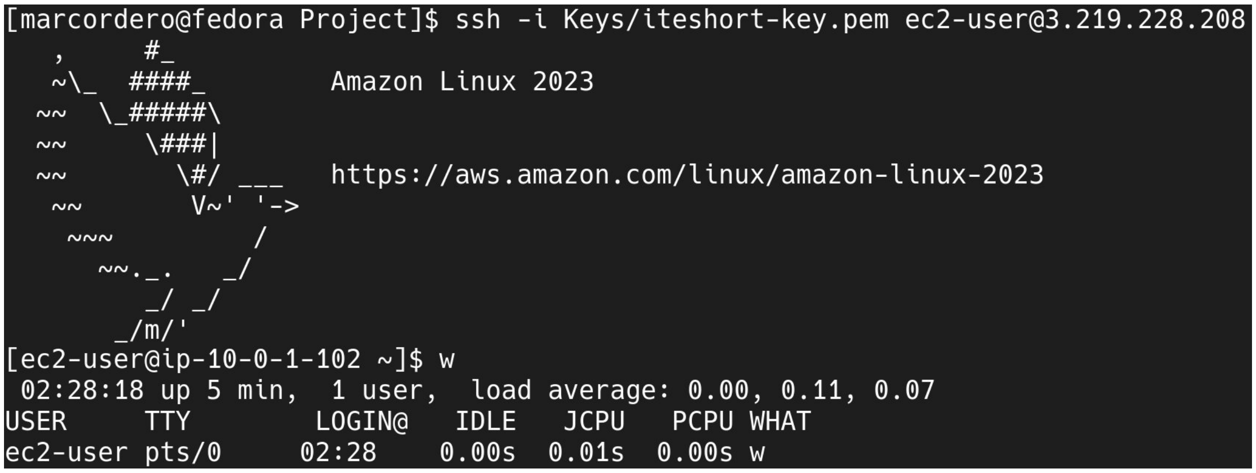


Name	Allocated IPv4 add...	Type	Allocation ID
web-ip	3.219.228.208	Public IP	eipalloc-0da02bf217f77576e



Elastic IP address associated successfully.
Elastic IP address 3.219.228.208 has been associated with instance i-010ed452975c5c611

SSH connection should be tested, as it will play an important role later on.



```
[marcordero@fedora Project]$ ssh -i Keys/iteshort-key.pem ec2-user@3.219.228.208
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-10-0-1-102 ~]$ w
 02:28:18 up 5 min,  1 user,  load average: 0.00, 0.11, 0.07
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
ec2-user pts/0     02:28    0.00s  0.01s  0.00s w
```

Once verified, the server's architecture will need to be present somewhere in the instance. To achieve this, two approaches can be followed: using *scp* to copy directly from a local machine to the instance, or, cloning a remote repository with the necessary code. The second method has more cloud in it than the first one, so it'll be preferred. This code can be found here (this is a hyperlink). **Note**: SSH keys and such configurations for remote repository access is omitted, but it's necessary to work with GitHub.

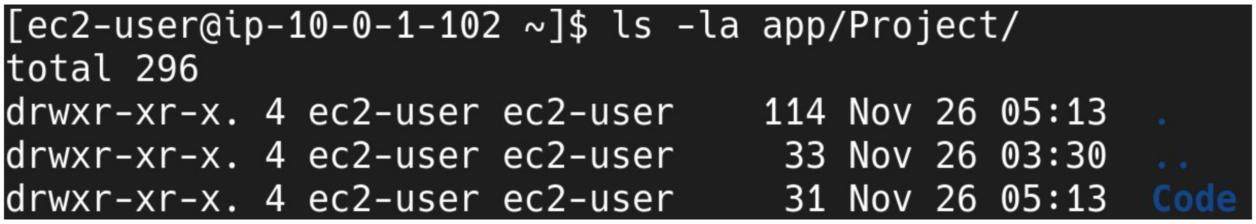
Although it's already been said that *scp* won't be used, it do needs to be used once for the server initial configuration. The file in bash extension format it's the following:

```bash
# Install dependencies
sudo dnf --assumeyes install git
sudo dnf --assumeyes install npm

# Create application container directory
mkdir /home/ec2-user/app
cd /home/ec2-user/app

# Fetch remote repository
git init
git remote add -f origin git@github.com:Marcox385/Cloud_O2023.git
git config core.sparseCheckout true
echo "Project/" >> .git/info/sparse-checkout
git pull origin main
```

To execute it, perform the following:

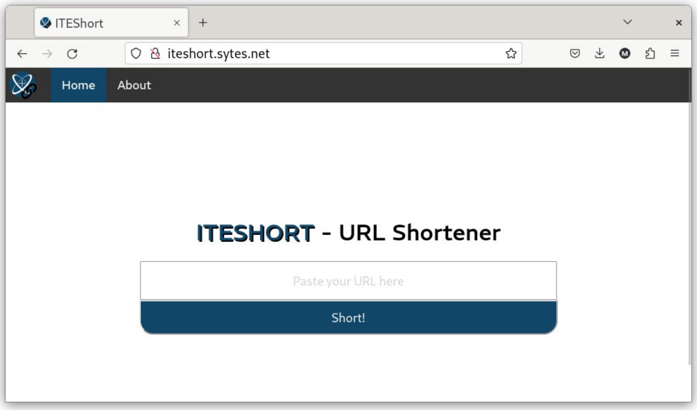
```
scp -i Keys/iteshort-key.pem Server/server.sh ec2-user@{given ip}:/home/ec2-
user/ && ssh -i Keys/iteshort-key.pem ec2-user@{given ip} “./server.sh”
```

Once all has been set, the application code should be inside */home/ec2-user/app* path.



```
[ec2-user@ip-10-0-1-102 ~]$ ls -la app/Project/
total 296
drwxr-xr-x. 4 ec2-user ec2-user    114 Nov 26 05:13  .
drwxr-xr-x. 4 ec2-user ec2-user     33 Nov 26 03:30  ..
drwxr-xr-x. 4 ec2-user ec2-user     31 Nov 26 05:13  Code
```

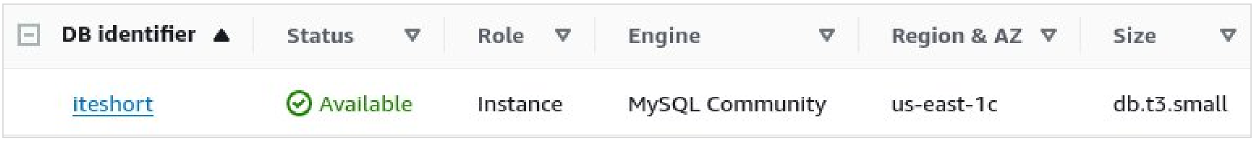
Applying the proper commands, the base application should now be running.



Services implementation

After the visual component of this infrastructure is available for testing, the next thing will be creating the RDS database with this configuration:

- Standard create
- Engine: MySQL – Community edition
- Engine Version: 8.0.33 or newest
- Templates: Dev/Test
- Availability and durability: Multi-AZ DB Instance
- Settings
    - Identifier: iteshort
    - Credentials: admin | adminpwd
- Instance configuration: burstable, db.t3.small
- Storage: 20GB gp3
- Connectivity
    - Don't connect to an EC2 resource
    - VPC: iteshort-vpc
    - Public access: Yes
    - New security group
        - Name: iteshort-db-sg (later on allow all connections to port 3306)
- Additional configuration
    - Initial database: iteshort



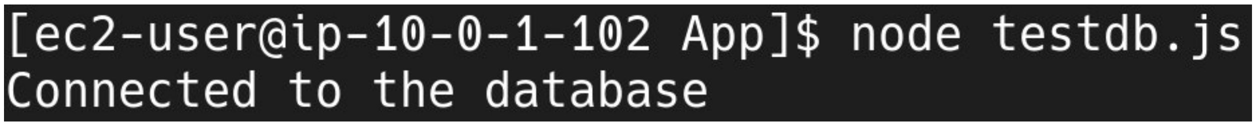
DB identifier	Status	Role	Engine	Region & AZ	Size
iteshort	Available	Instance	MySQL Community	us-east-1c	db.t3.small

The main table schema would look like the following:

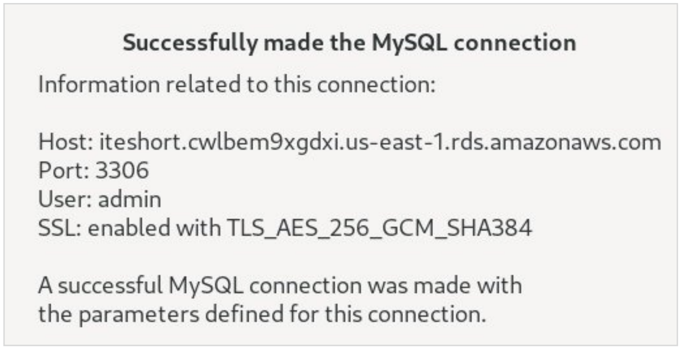
Shortened_urls	
url_hash	String, PK
original_url	String, SK
short_url	String
count	Number
last_access	String

This should be self descriptive, but in case is not clear enough, the original URL will be hashed to be used as the primary or partition key, original URL will contain the raw URL string and it'll be useful for fast search, short url will have the new shortened format of the resource, count will hold the amount of times that URL was visited, and last access will indicate the last time the URL was accessed.

An RDS Proxy would be ideal for Lambda connection, but it couldn't be used at the moment of this development. Connectivity should also be tested to verify normal operation.



```
[ec2-user@ip-10-0-1-102 App]$ node testdb.js
Connected to the database
```



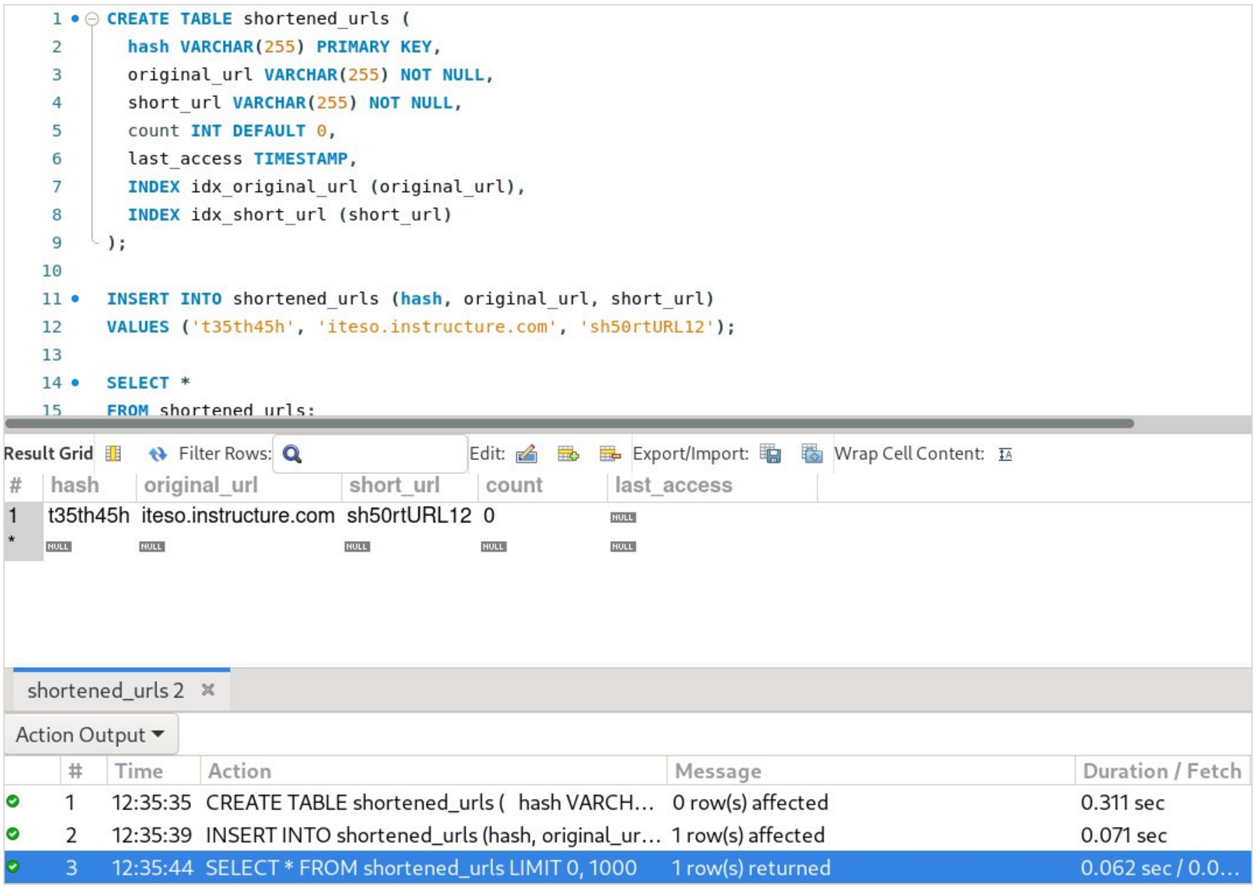
Successfully made the MySQL connection

Information related to this connection:

Host: iteshort.cwlbem9xgdxi.us-east-1.rds.amazonaws.com
Port: 3306
User: admin
SSL: enabled with TLS_AES_256_GCM_SHA384

A successful MySQL connection was made with the parameters defined for this connection.

Using MySQL Workbench, the schema definition and insert testing can be done easily.



```sql
CREATE TABLE shortened_urls (
  hash VARCHAR(255) PRIMARY KEY,
  original_url VARCHAR(255) NOT NULL,
  short_url VARCHAR(255) NOT NULL,
  count INT DEFAULT 0,
  last_access TIMESTAMP,
  INDEX idx_original_url (original_url),
  INDEX idx_short_url (short_url)
);

INSERT INTO shortened_urls (hash, original_url, short_url)
VALUES ('t35th45h', 'iteso.instructure.com', 'sh50rtURL12');

SELECT *
FROM shortened_urls;
```

#	hash	original_url	short_url	count	last_access
1	t35th45h	iteso.instructure.com	sh50rtURL12	0	NULL
*	NULL	NULL	NULL	NULL	NULL

	#	Time	Action	Message	Duration / Fetch
✓	1	12:35:35	CREATE TABLE shortened_urls (hash VARCH…	0 row(s) affected	0.311 sec
✓	2	12:35:39	INSERT INTO shortened_urls (hash, original_ur…	1 row(s) affected	0.071 sec
✓	3	12:35:44	SELECT * FROM shortened_urls LIMIT 0, 1000	1 row(s) returned	0.062 sec / 0.0…

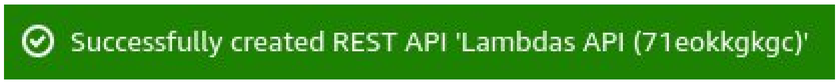
Having the database ready for writing, one of the last steps will be creating the lambda functions that will act as proxies between the application and RDS. Create Lambdas with names "hashURL", "removeHead", "shortenURL", all with these configurations

- Runtime: Python 3.11
- Default architecture
- Execution role
  - Existing role – LabRole
  - Create a new role for real implementations
- Advanced settings
  - Enable function URL
    - Auth type: NONE
    - Invoke mode: BUFFERED
    - Configure CORS
    - Enable VPC: Select all network resources previously created

The creation of these Lambdas will follow a microservice architecture, in which several components of the same design collaborate between them to achieve a greater goal.

Lambda natively usage or through endpoints might result in several complications, so, to avoid this, API Gateway will be used to create a proxy between services:

- Build REST API
- New API
- API name: Lambdas API
- Endpoint type: Regional

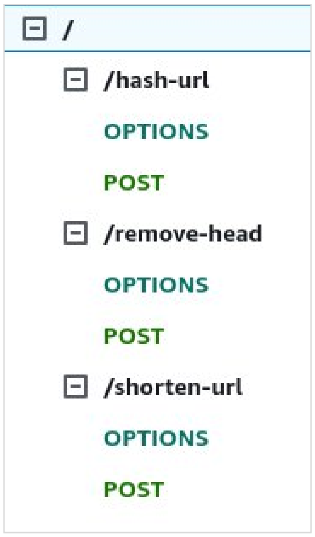

Successfully created REST API 'Lambdas API (71eokkgkgc)'

After the API has been created, search for "Create resource" to add the following:

- hash-url
- remove-head
- shorten-url

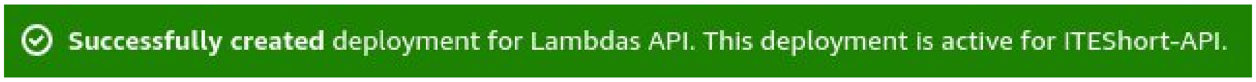
With these requirements:

- Method: POST
- Integration: Lambda function
  - us-east-1
  - Select corresponding Lambda function
- Default timeout


/
/hash-url
OPTIONS
POST
/remove-head
OPTIONS
POST
/shorten-url
OPTIONS
POST

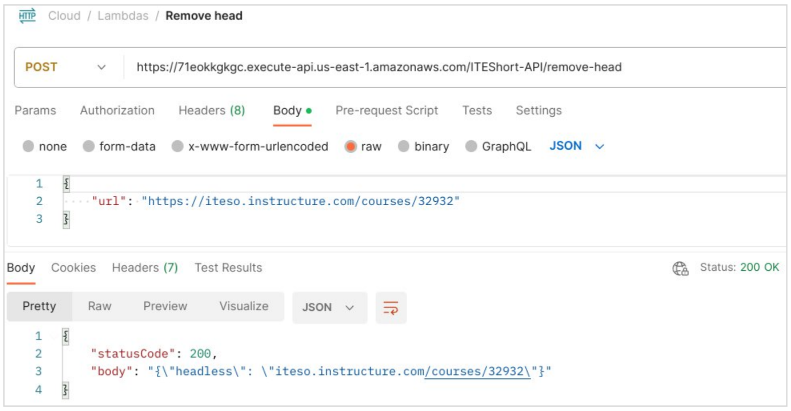
After all resources have been added, it should look like the image on the right.

Once all is added, deploy the API with stage name “ITEShort-API”.



Successfully created deployment for Lambdas API. This deployment is active for ITEShort-API.

API gateway will provide a URL for this API. All actions can be tested through any HTTP client such as Postman.

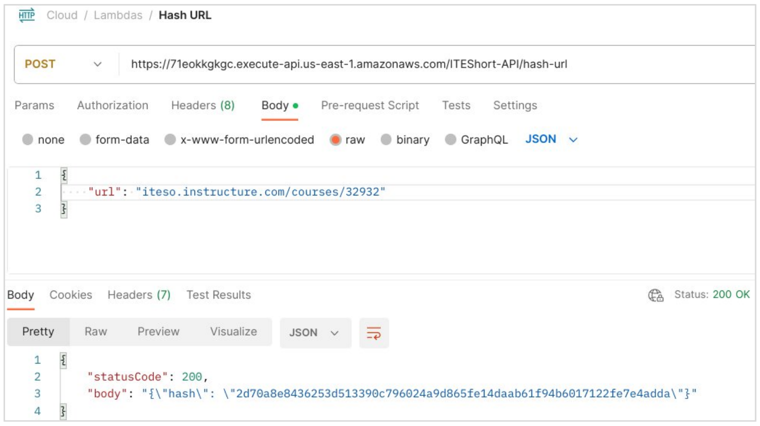

Cloud / Lambdas / Remove head
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON

POST https://71eokkgkgc.execute-api.us-east-1.amazonaws.com/ITEShort-API/remove-head

```
{
    "url": "https://iteso.instructure.com/courses/32932"
}
```

Status: 200 OK

```
{
    "statusCode": 200,
    "body": "{\"headless\": \"iteso.instructure.com/courses/32932\"}"
}
```

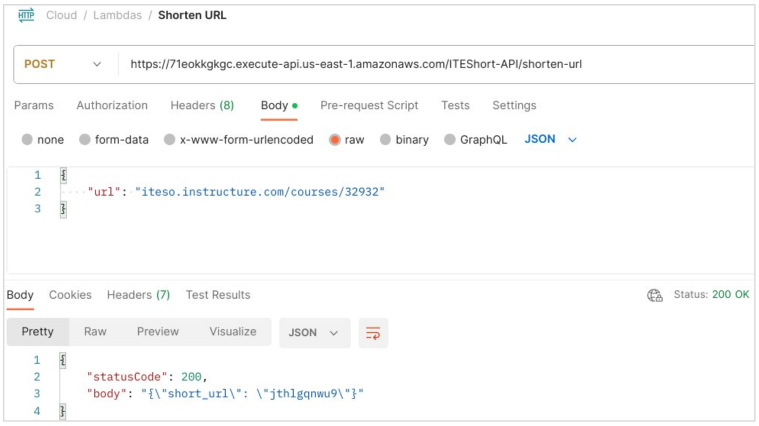

Cloud / Lambdas / Hash URL
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON

POST https://71eokkgkgc.execute-api.us-east-1.amazonaws.com/ITEShort-API/hash-url

```
{
    "url": "iteso.instructure.com/courses/32932"
}
```

Status: 200 OK

```
{
    "statusCode": 200,
    "body": "{\"hash\": \"2d70a8e8436253d513390c796024a9d865fe14daab61f94b6017122fe7e4adda\"}"
}
```


Cloud / Lambdas / Shorten URL
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON

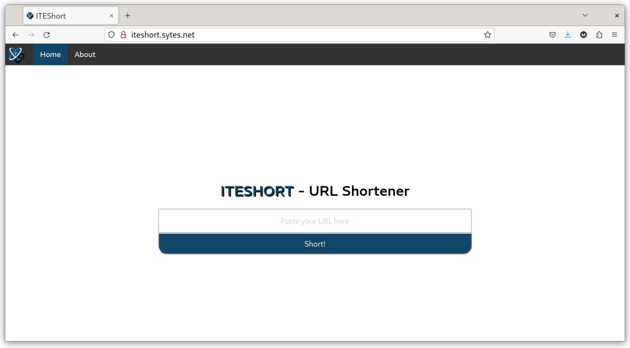
POST https://71eokkgkgc.execute-api.us-east-1.amazonaws.com/ITEShort-API/shorten-url

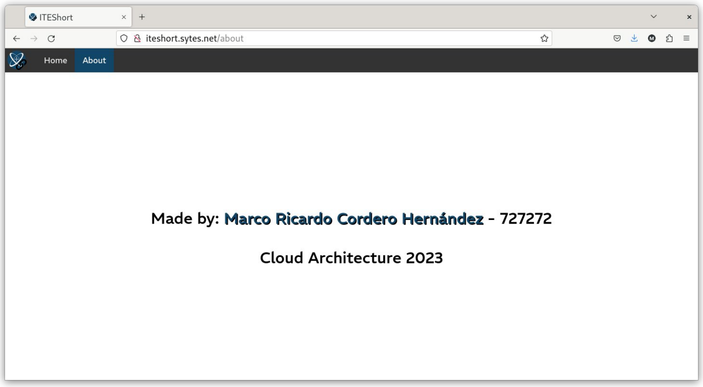
```
{
    "url": "iteso.instructure.com/courses/32932"
}
```

Status: 200 OK

```
{
    "statusCode": 200,
    "body": "{\"short_url\": \"jthlgqnwu9\"}"
}
```

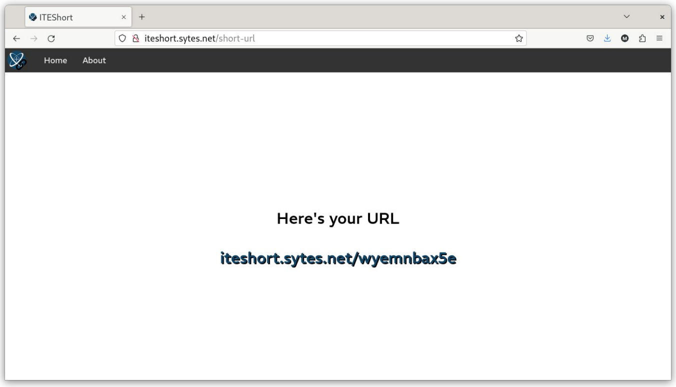
By this point, and taking into account that all the necessary logic to achieve the app’s purpose it’s inside the EC2 instances and the Lambdas, the app should now be running and being able to short URLs.





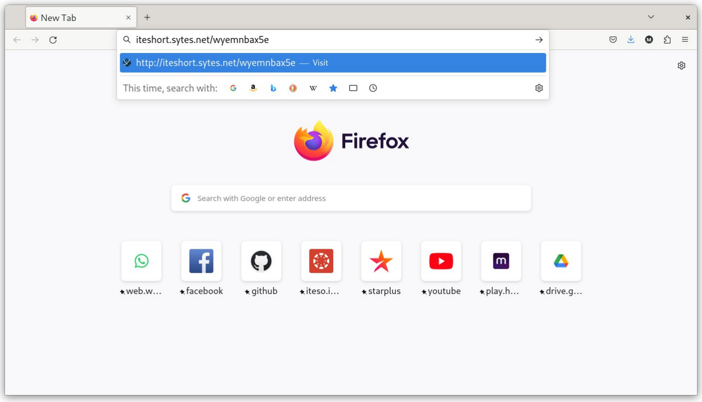
Made by: Marco Ricardo Cordero Hernández - 727272

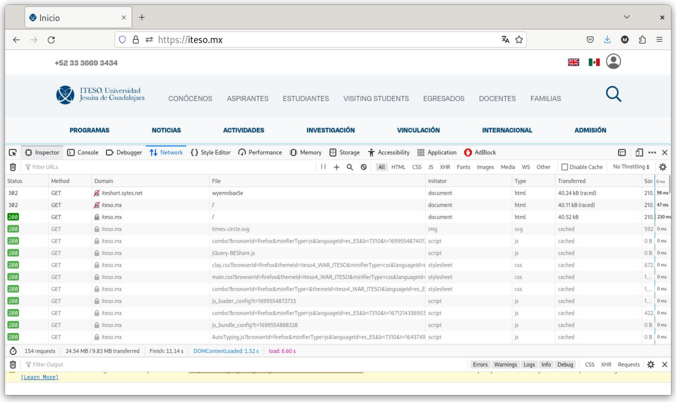
Cloud Architecture 2023



Here's your URL

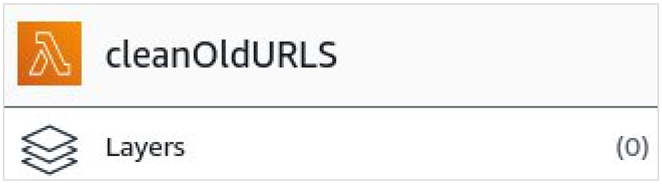
iteshort.sytes.net/wyemnbax5e


New Tab
iteshort.sytes.net/wyemnbax5e
http://iteshort.sytes.net/wyemnbax5e — Visit
This time, search with:
Firefox
Search with Google or enter address
web.w...
facebook
github
iteso.i...
starplus
youtube
play.h...
drive.g...

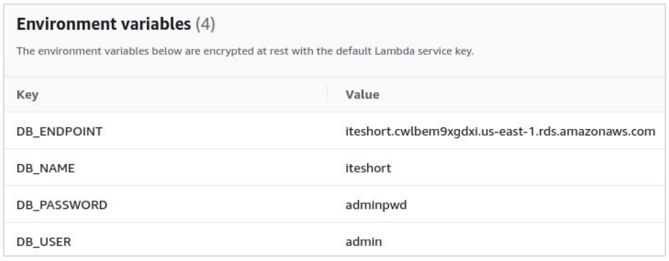


Status	Method	Domain	File	Initiator	Type	Transferred
302	GET	iteshort.sytes.net	wyemnbax5e	document	html	40.24 kB (raced)
302	GET	iteso.mx	/	document	html	40.11 kB (raced)
200	GET	iteso.mx	/	document	html	40.52 kB

The last thing to do is to create an additional Lambda function to periodically delete non used URLs; it will have the same settings as before. **Note**: the libraries required for this function have to be installed locally and then uploaded as a .zip or accessed via an S3 bucket into the function, as Lambda doesn't provide command line utilities.



Once created, set the environment variables inside configuration.



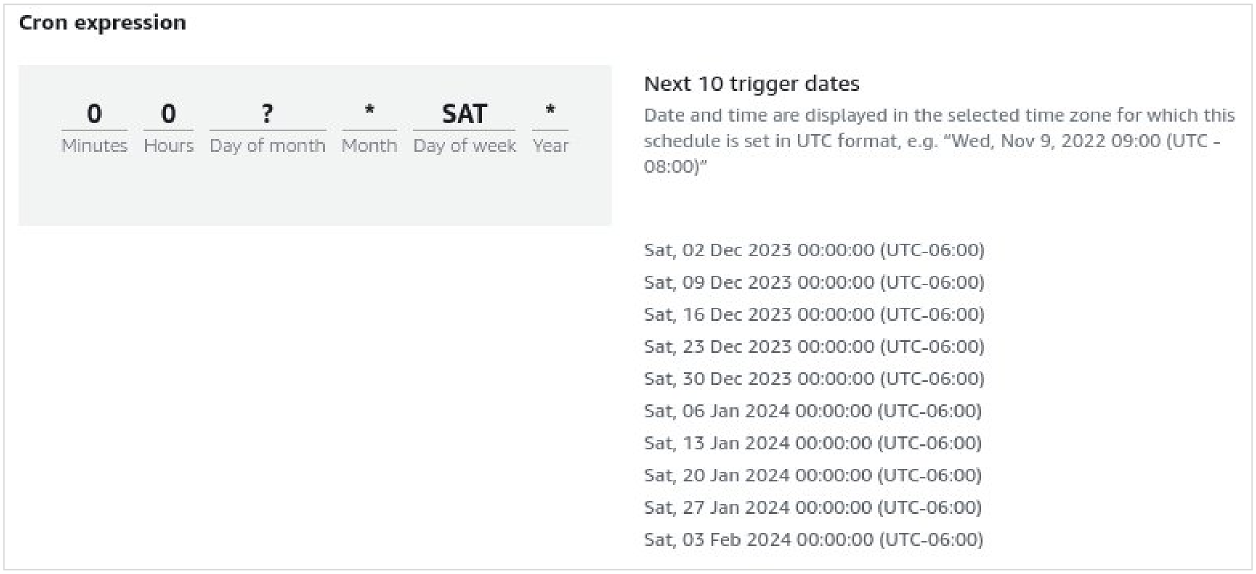
Environment variables (4)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
DB_ENDPOINT	iteshort.cwlbem9xgdxi.us-east-1.rds.amazonaws.com
DB_NAME	iteshort
DB_PASSWORD	adminpwd
DB_USER	admin

After proper testing has been conducted, create an EventBridge schedule that runs every sunday at midnight (this can be done through cron expressions). Inside, set the following:

- Name: clean_database
- Pattern
  - Ocurrence: Recurring
  - Type: Cron-based (0 0 ? * SAT *)
  - Flexible time window: Off
- Timeframe
  - Time zone: UTC-06:00
- Target detail
  - Templated targets > Lambda > Select cleanOldURLS (without payload)
- Schedule state: Enable
- Action after completion: NONE

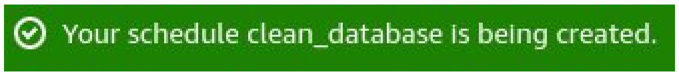

Cron expression

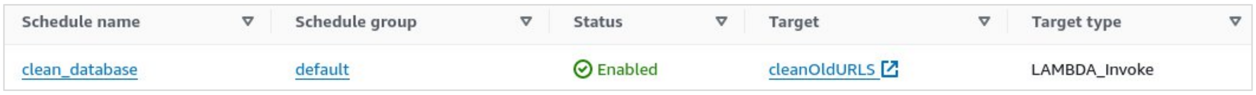
Minutes	Hours	Day of month	Month	Day of week	Year
0	0	?	*	SAT	*

Next 10 trigger dates

Date and time are displayed in the selected time zone for which this schedule is set in UTC format, e.g. "Wed, Nov 9, 2022 09:00 (UTC - 08:00)"

```
Sat, 02 Dec 2023 00:00:00 (UTC-06:00)
Sat, 09 Dec 2023 00:00:00 (UTC-06:00)
Sat, 16 Dec 2023 00:00:00 (UTC-06:00)
Sat, 23 Dec 2023 00:00:00 (UTC-06:00)
Sat, 30 Dec 2023 00:00:00 (UTC-06:00)
Sat, 06 Jan 2024 00:00:00 (UTC-06:00)
Sat, 13 Jan 2024 00:00:00 (UTC-06:00)
Sat, 20 Jan 2024 00:00:00 (UTC-06:00)
Sat, 27 Jan 2024 00:00:00 (UTC-06:00)
Sat, 03 Feb 2024 00:00:00 (UTC-06:00)
```


Your schedule clean_database is being created.



Schedule name	Schedule group	Status	Target	Target type
clean_database	default	✓ Enabled	cleanOldURLS	LAMBDA_Invoke

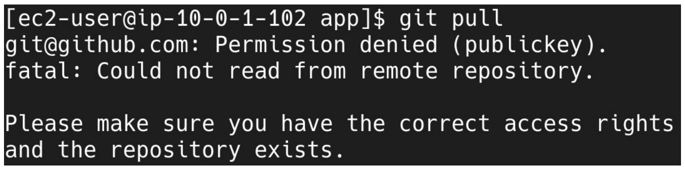
This marks the end of the technical part of the project.

Problems and solutions

Several problems arose through this development, but most of them where quickly spotted by prior experience and pain of repetition. Most of these were network related, in which inbound and outbound rules were not set correctly, and thus required traffic wasn't reaching its destination.

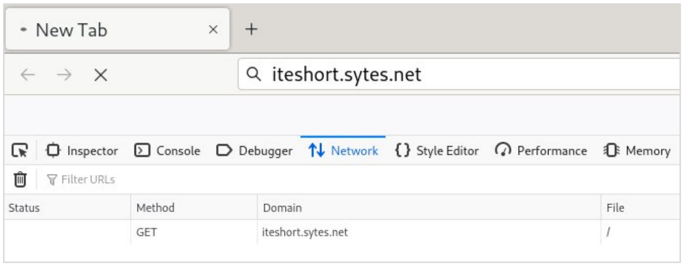
One big problem was the use of the database, because before using RDS, DynamoDB was set to be used as the default database. At the end, the implemented solution is still serverless, however, DynamoDB is far more straightforward than any RDS configuration, however, connection was giving plenty of troubles that just kept taking invaluable time back from the whole development, thus, the ultimate solution to this persistent problem was switching to MySQL.

Another odd behavior is found when reconnecting to the instance through SSH and trying to pull the remote repository containing the code.



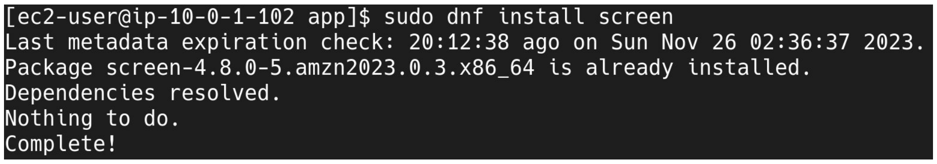
```
[ec2-user@ip-10-0-1-102 app]$ git pull
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

A related problem has to do with the express server sudden termination, making seem that the application it's accessible forever, but after some time, it won't respond.



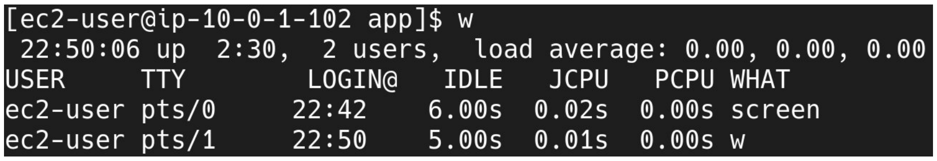
Status	Method	Domain	File
	GET	iteshort.sytes.net	/

Both of these are solved by the *screen* utility inside the EC2 instance, as this tool acts as a multiplexer, thus allowing to perform long running tasks. It's suitable because it's already installed in the instance



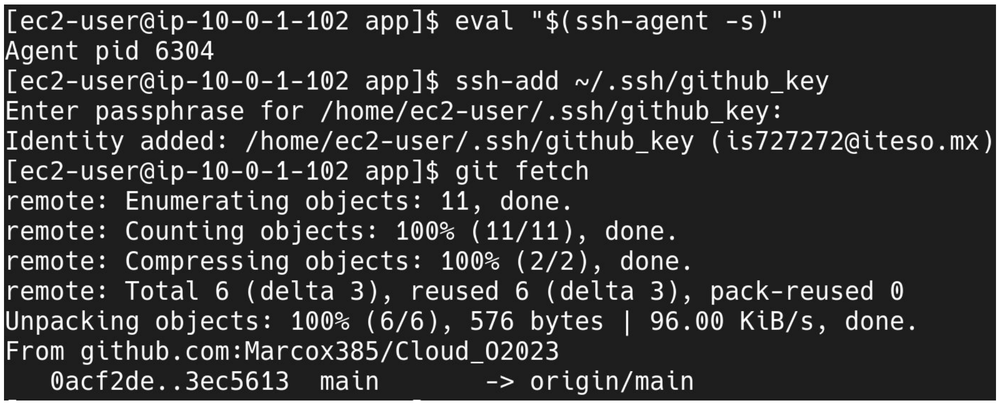
```
[ec2-user@ip-10-0-1-102 app]$ sudo dnf install screen
Last metadata expiration check: 20:12:38 ago on Sun Nov 26 02:36:37 2023.
Package screen-4.8.0-5.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

By simply using the mentioned command, a new persistent session will be enabled



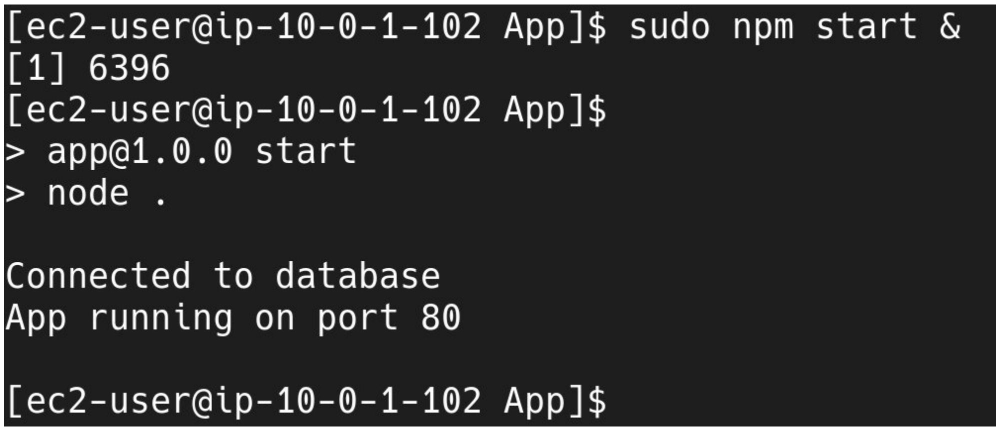
```
[ec2-user@ip-10-0-1-102 app]$ w
 22:50:06 up  2:30,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
ec2-user pts/0     22:42    6.00s  0.02s  0.00s screen
ec2-user pts/1     22:50    5.00s  0.01s  0.00s w
```

Now, to solve the first problem, the SSH has to be running and the SSH key created has to be added. The benefit of using the screen utility is that not only the server won't timeout the connection, but it will also persist the running tasks inside it even if an external exit is used.



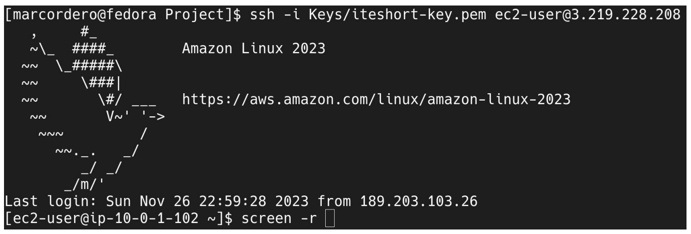
```
[ec2-user@ip-10-0-1-102 app]$ eval "$(ssh-agent -s)"
Agent pid 6304
[ec2-user@ip-10-0-1-102 app]$ ssh-add ~/.ssh/github_key
Enter passphrase for /home/ec2-user/.ssh/github_key:
Identity added: /home/ec2-user/.ssh/github_key (is727272@iteso.mx)
[ec2-user@ip-10-0-1-102 app]$ git fetch
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 3), reused 6 (delta 3), pack-reused 0
Unpacking objects: 100% (6/6), 576 bytes | 96.00 KiB/s, done.
From github.com:Marcox385/Cloud_O2023
   0acf2de..3ec5613  main       -> origin/main
```

For the second problem, the usual starting command can be used, even sending it as a background process, the difference is that screen will keep the execution at least until the instance is rebooted or shutdown.

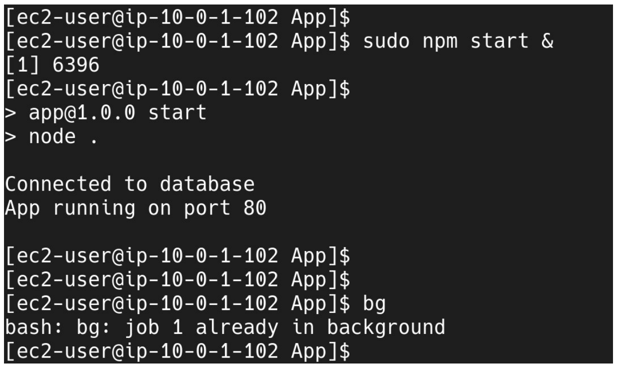


```
[ec2-user@ip-10-0-1-102 App]$ sudo npm start &
[1] 6396
[ec2-user@ip-10-0-1-102 App]$
> app@1.0.0 start
> node .

Connected to database
App running on port 80

[ec2-user@ip-10-0-1-102 App]$
```

If the client console it's closed, the SSH will be lost, however, when reconnecting, using *mirror -r* will resume its execution.



```
[marcordero@fedora Project]$ ssh -i Keys/iteshort-key.pem ec2-user@3.219.228.208
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sun Nov 26 22:59:28 2023 from 189.203.103.26
[ec2-user@ip-10-0-1-102 ~]$ screen -r
```



```
[ec2-user@ip-10-0-1-102 App]$
[ec2-user@ip-10-0-1-102 App]$ sudo npm start &
[1] 6396
[ec2-user@ip-10-0-1-102 App]$
> app@1.0.0 start
> node .

Connected to database
App running on port 80

[ec2-user@ip-10-0-1-102 App]$
[ec2-user@ip-10-0-1-102 App]$
[ec2-user@ip-10-0-1-102 App]$ bg
bash: bg: job 1 already in background
[ec2-user@ip-10-0-1-102 App]$
```

This covers all the problem found while developing the solution.

Experiments and results

As in pasts practices and implementations, the current project can also be seen as the biggest experiment as of today, or at least in this course, because no instructions were defined and no clear path was set to build what's been show. The experiment began months prior to writing a single line of code, when the goal and objectives of the whole were being defined and the technologies or services were also being analyzed in order to determine what was best for the proposal.

If the abstraction level is taken to the maximum, the whole class was a big experiment in which several results in the form of reports, such as the current one, were presented. All of that developments aided not only in the final results for this project, but also in the growth of personal abilities and vast knowledge acquirement that will surely be useful in further experiments.

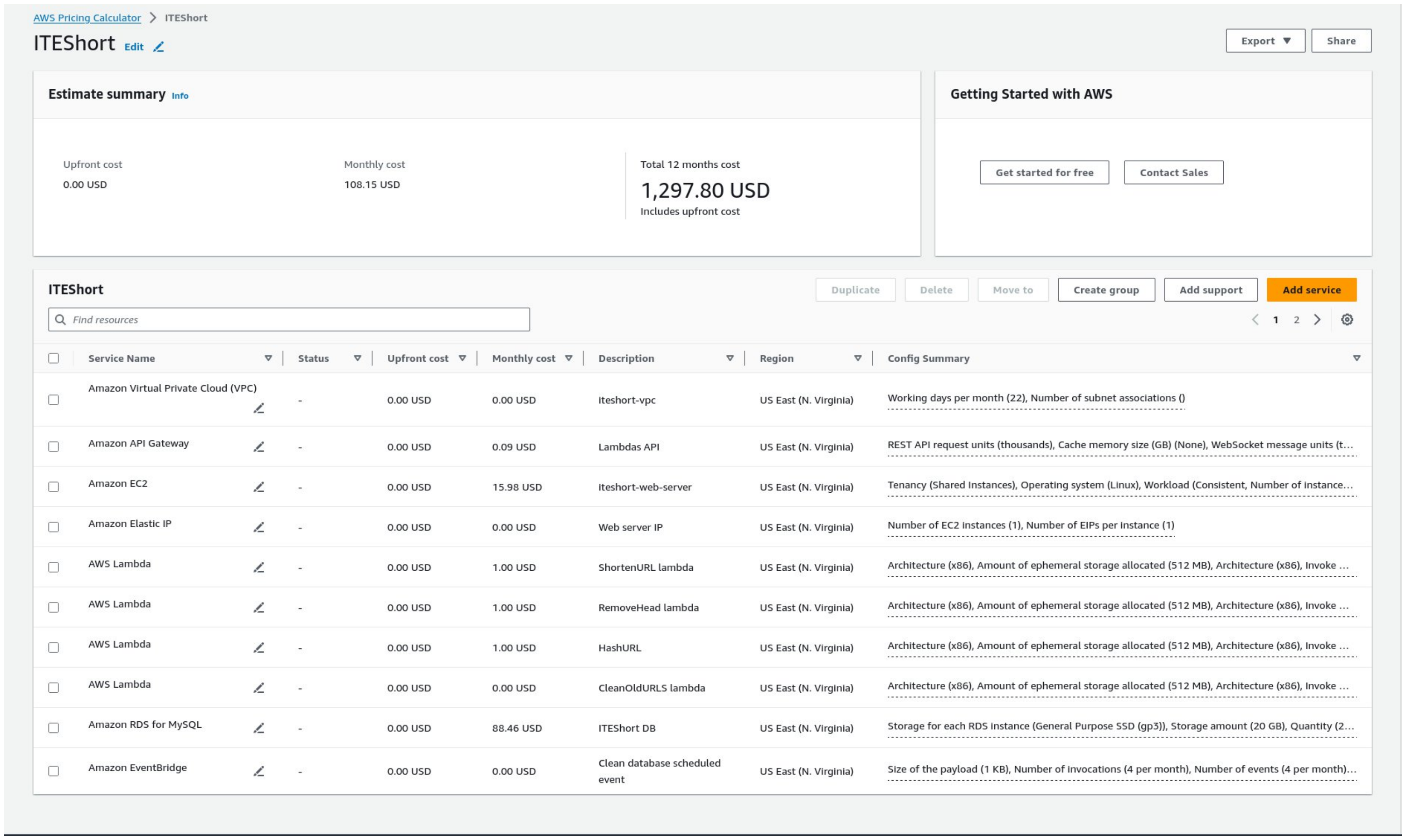
Other than the discovery of previously unseen services and their implementations, no experiments were conducted. Perhaps taking the human body to the limits by sleeping 2 hours in 3 days, but that goes far beyond the current scope.

Cost analysis

Although it might seem that a lot was consumed, in reality, it's a pretty acceptable amount of resources, and this being:

– Single VPC
– API Gateway with 25 thousand REST API requests per month
– EC2 instance type t3.small with 100% utilization per month and 8Gb gp2 storage
– One elastic IP
– Multiple lambda functions with approximately 5 million requests per month
– A single lambda function invoked a maximum of four times per month
– Two MySQL RDS with 60 hours utilization a week on Multi-AZ
– One EventBridge schedule
– One non-implemented Route 53 domain

Having this services, the operational costs of a real world maintenance of this project would be:



ITEShort

Estimate summary Info

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	108.15 USD	1,297.80 USD Includes upfront cost

ITEShort

Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config Summary
Amazon Virtual Private Cloud (VPC)	-	0.00 USD	0.00 USD	iteshort-vpc	US East (N. Virginia)	Working days per month (22), Number of subnet associations ()
Amazon API Gateway	-	0.00 USD	0.09 USD	Lambdas API	US East (N. Virginia)	REST API request units (thousands), Cache memory size (GB) (None), WebSocket message units (t...
Amazon EC2	-	0.00 USD	15.98 USD	iteshort-web-server	US East (N. Virginia)	Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instance...
Amazon Elastic IP	-	0.00 USD	0.00 USD	Web server IP	US East (N. Virginia)	Number of EC2 instances (1), Number of EIPs per instance (1)
AWS Lambda	-	0.00 USD	1.00 USD	ShortenURL lambda	US East (N. Virginia)	Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke ...
AWS Lambda	-	0.00 USD	1.00 USD	RemoveHead lambda	US East (N. Virginia)	Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke ...
AWS Lambda	-	0.00 USD	1.00 USD	HashURL	US East (N. Virginia)	Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke ...
AWS Lambda	-	0.00 USD	0.00 USD	CleanOldURLS lambda	US East (N. Virginia)	Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke ...
Amazon RDS for MySQL	-	0.00 USD	88.46 USD	ITEShort DB	US East (N. Virginia)	Storage for each RDS instance (General Purpose SSD (gp3)), Storage amount (20 GB), Quantity (2...
Amazon EventBridge	-	0.00 USD	0.00 USD	Clean database scheduled event	US East (N. Virginia)	Size of the payload (1 KB), Number of invocations (4 per month), Number of events (4 per month)...

Conclusions

Having implemented something so challenging (at least for a single person) can't result in nothing bad, as the ultimate gain obtained from this is non other than experience. Maybe the near career path won't be focusing on cloud architecture or anything remotely similar, but as already stated, when it comes to knowledge and exploring of the current technology industry, nothing comes as too much.

Perhaps one interesting challenge left as a far away exercise would be implementing the exact same proposal on different clouds, because there is no secret that there was a preference for AWS, this might be because of the fact that the resources that made possible all the steps that led to this conclusion were made available by the same provider. The thing is that a disjunctive forms from this fact: there's the side that would say that learning and working with AWS is a great opportunity to develop talent in a world in which their cloud is predominant, and, the opposite that would say that one can't depend solemnly on a single technology (or set of them) and that monopolies aren't a good thing. The ambivalence found in this dilemma can be rather challenging to address, as both parts have some level of logic and truth in them, however, when a technology goes into vogue, even if is not that appealing or easy to implement and migrate to, big corporations will start to demand it in resumes, as far as to asking for it just because.

As many other aspects found inside this project, the questioning of whether AWS is subjectively good or bad is beyond the current goals, and these are already satisfied because the proposal made a while ago sees the light on the demonstrated development. The fact that such demonstration is possible not only overlooks the technological moral dilemma discussed before, but also serves the double purpose of demonstrating that all the contents seen throughout the class were useful and they taught valuable lessons, not only in the academic aspect, but also in soft skills such as resiliency, because the fact that a compendium of tools are popular doesn't exempt them from having the most nefarious, obscure, and devilish problems; surely they made themselves clear in every implementation step.

Many words have already been said, and many more could surely come, but for now, is time to give an end to this hectic project with the reassurance that all of its theory, implementation, problems, planning, developing, documentation, and every other aspect found within will lead to a better professional profile. And if that wasn't the main goal of this whole course, well, then maybe all these is devoid of value after all. Let's hope is not like that.

Thank you.

References

[1] Board of Regents of the University System of Georgia. 'A Brief History of the Internet'. [Online]. Available: https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml.

[2] Z. Bloom. 'The History of the URL'. [Online]. Available: https://blog.cloudflare.com/the-history-of-the-url/.

[3] MDN Web Docs. 'World Wide Web'. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/World_Wide_Web.

[4] MDN Web Docs. 'What are hyperlinks?'. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_are_hyperlinks.

[5] S. Mathi. 'How The URL Was Built'. [Online]. Available: https://www.welcometothejungle.com/en/articles/btc-url-internet.

[6] Rebrandly. 'The Entire History of URL Shorteners'. [Online]. Available: https://rebrandly.com/blog/link-management/the-history-of-url-shorteners/.

[7] A. Cox. 'QR Code Size: The Ultimate QR Sizing Guide'. [Online]. Available: https://tritonstore.com.au/qr-code-size/.

[8] Aws.amazon.com. 'Cloud Computing Services'. [Online]. Available: https://aws.amazon.com/.

[9] Aws.amazon.com. 'Amazon Route 53'. [Online]. Available: https://aws.amazon.com/route53/.

[10] Aws.amazon.com. 'Amazon API Gateway'. [Online]. Available: https://aws.amazon.com/api-gateway/.

[11] Aws.amazon.com. 'Connect to the internet using an internet gateway'. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html.

[12] Aws.amazon.com. 'Amazon EC2'. [Online]. Available: https://aws.amazon.com/ec2/.

[13] Aws.amazon.com. 'AWS Lambda'. [Online]. Available: https://aws.amazon.com/lambda/.

[14] Aws.amazon.com. 'Amazon RDS for MySQL'. [Online]. Available: https://aws.amazon.com/rds/mysql/.

[15] Aws.amazon.com. 'Amazon EventBridge'. [Online]. Available: https://aws.amazon.com/eventbridge/.

[16] Aws.amazon.com. ‘Elastic IP addresses’. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html.

[17] Aws.amazon.com. ‘Amazon Virtual Private Cloud’. [Online]. Available: https://aws.amazon.com/vpc/.

[18] Aws.amazon.com. ‘Regions and Availability Zones’. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

[19] Aws.amazon.com. ‘Subnets for your VPC’. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html.