# Complaint’s System – 3rd Parties – Part 2 - AWS

## Introduction

Our goal today would be to integrate AWS(s3) in our application. For this purpose, we will use “boto3” library.

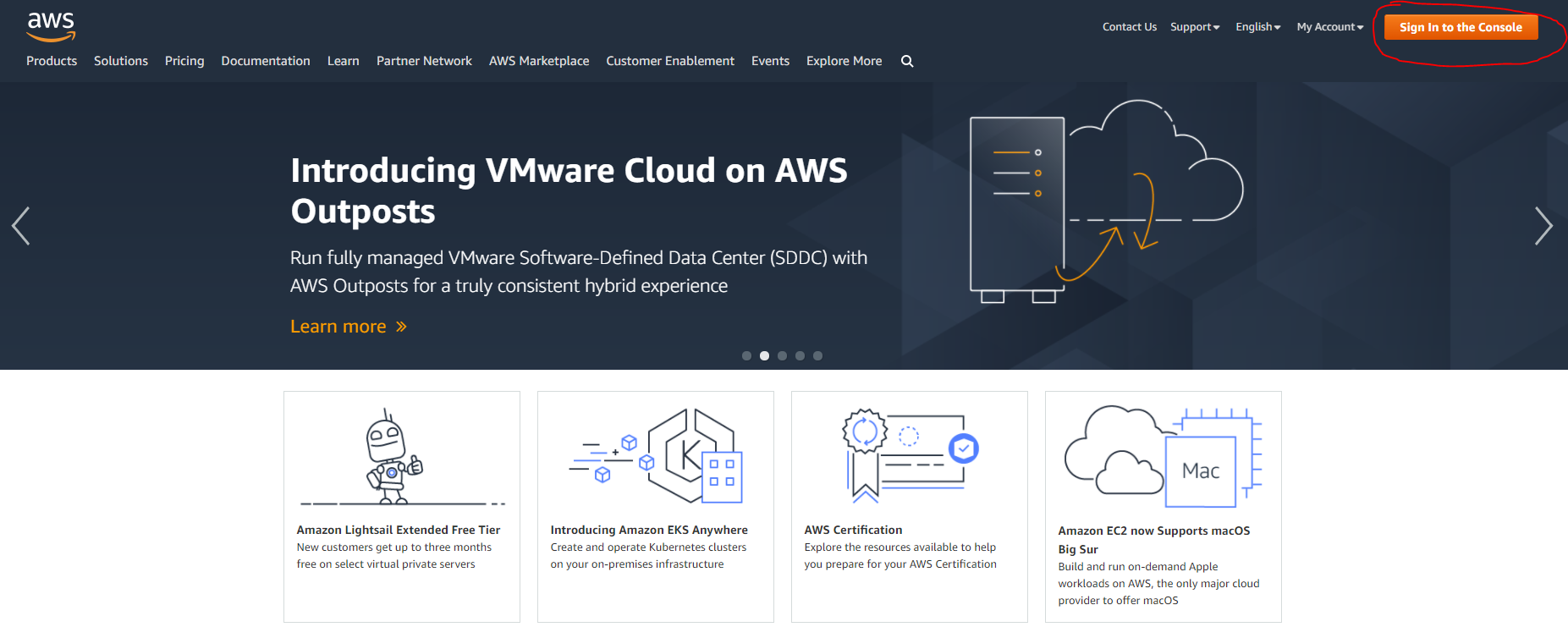
First, we will start with S3 by installing boto3 and connecting our account. Then we will upload the photos from complainers for their expenses. Also, we will use s3 to store our approvers' certificates.

We will continue to develop our architecture and rely on clean code and structured files.

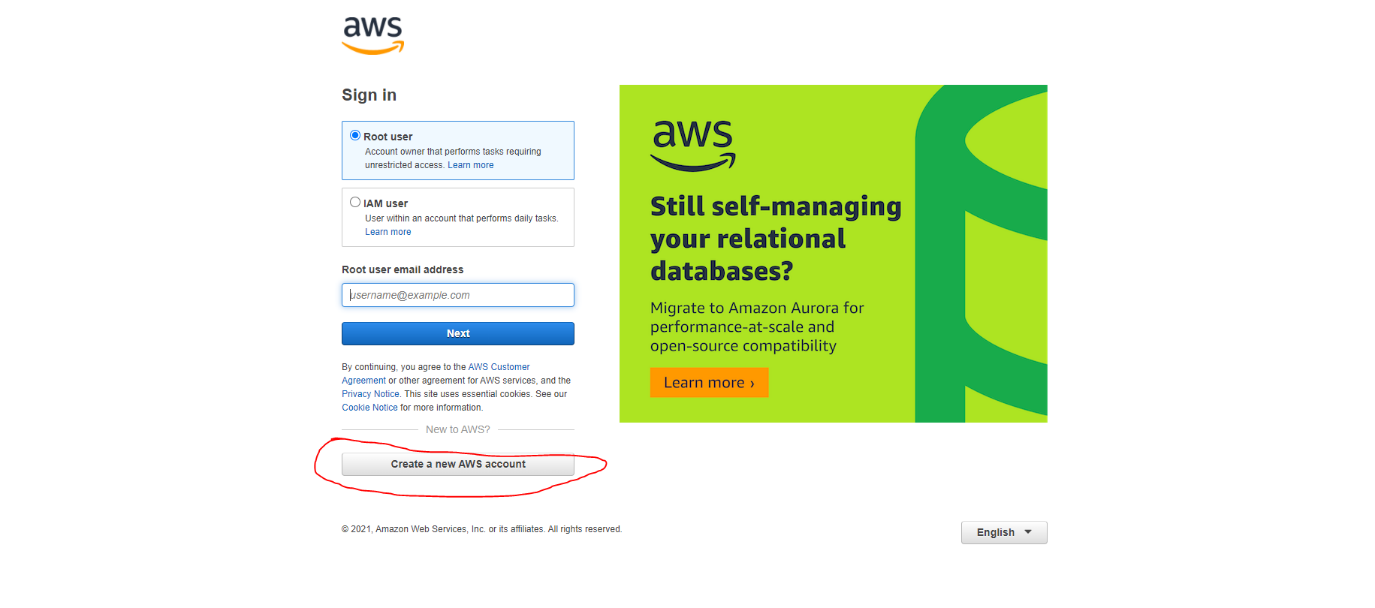
## Set up AWS account

First, we need to set up an AWS account. You can skip this step if you already have one.

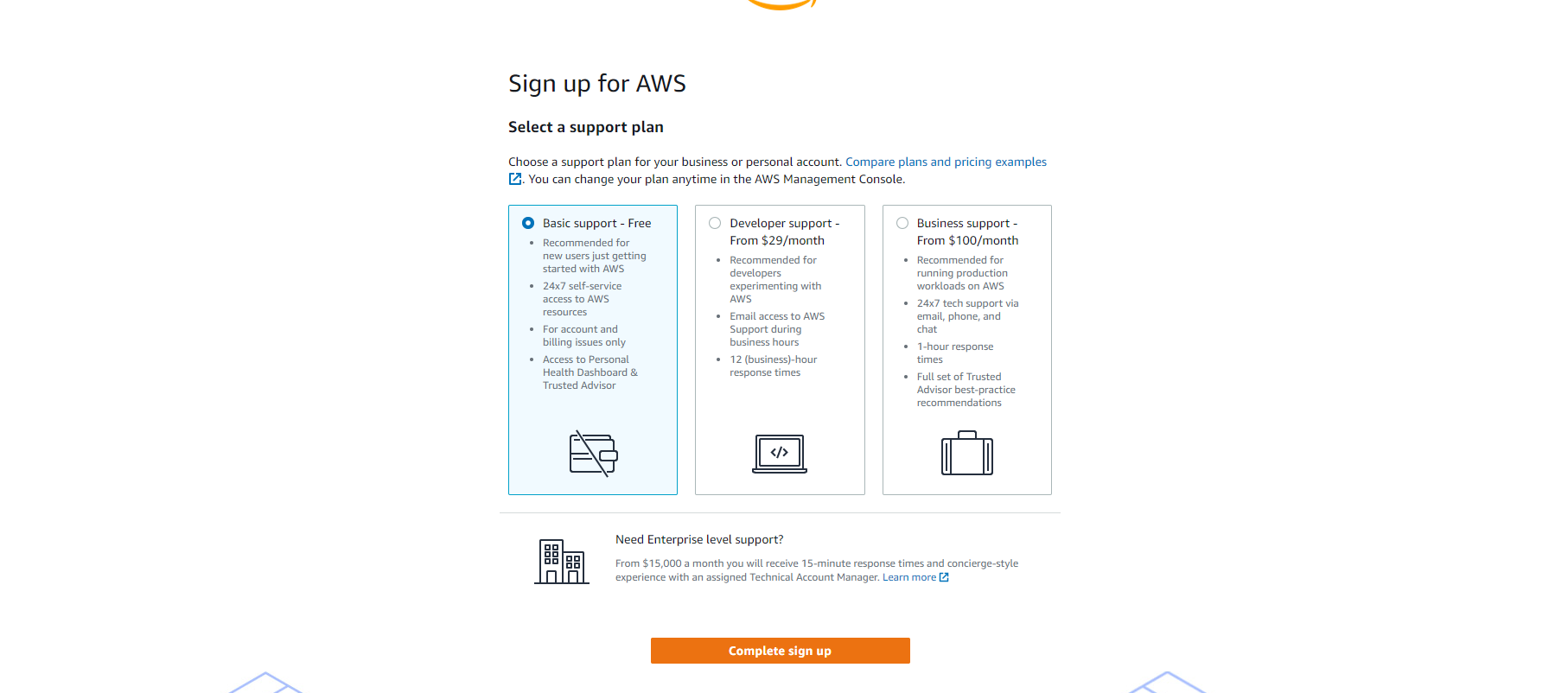
Go to <https://aws.amazon.com/> and click the **Sign in to the Console**

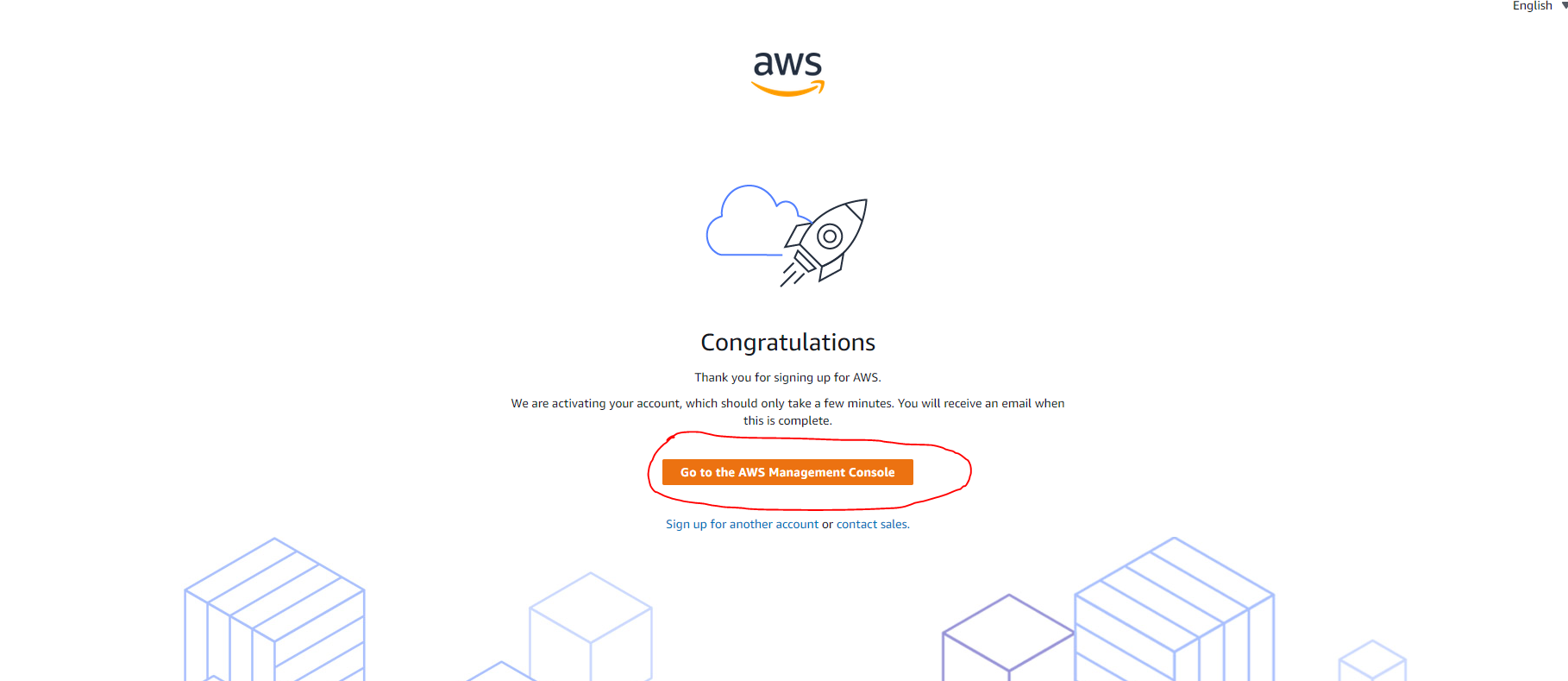


Then **Create a new AWS account**



For security reasons, we can not show the whole registration process because sensitive information would be exposed. You have to follow the 5 steps registration process. AWS will charge you 1 USD dollar for the registration. Next, you have to verify your phone number by entering the 4 digit code they will send you as the last step. When you finish the registration process successfully you will see something similar to:

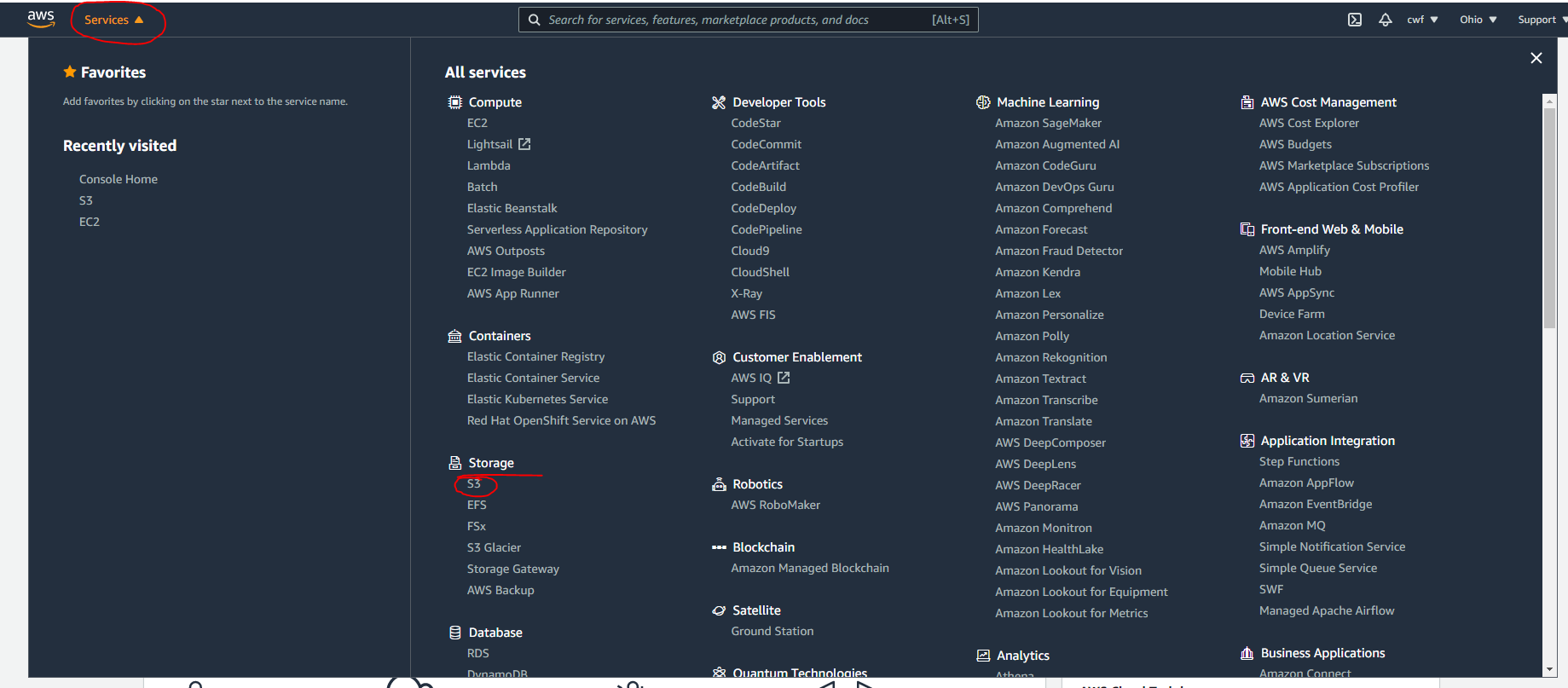


Choose the basic plan, so that you do not get any additional charges. Click **Complete sign up** 

And then Go to the **AWS Management Console**.

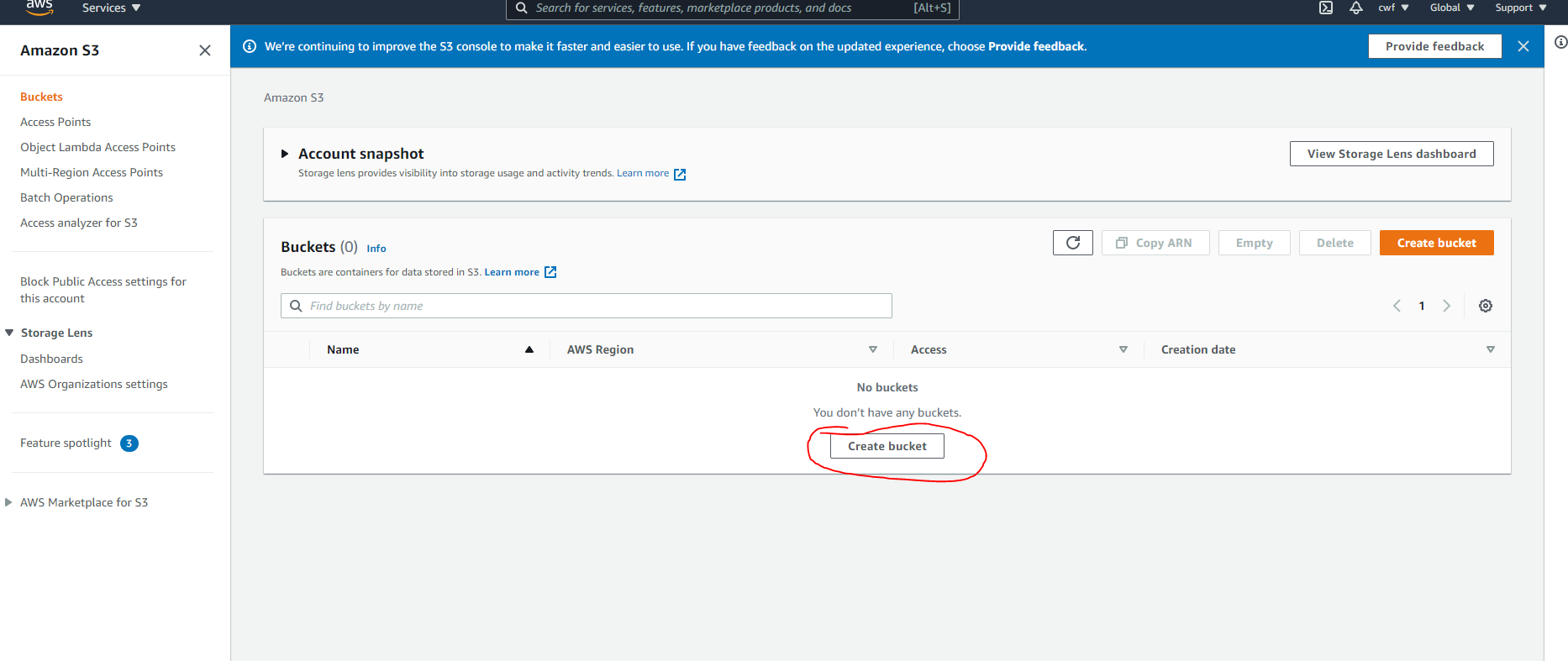
## Set up S3

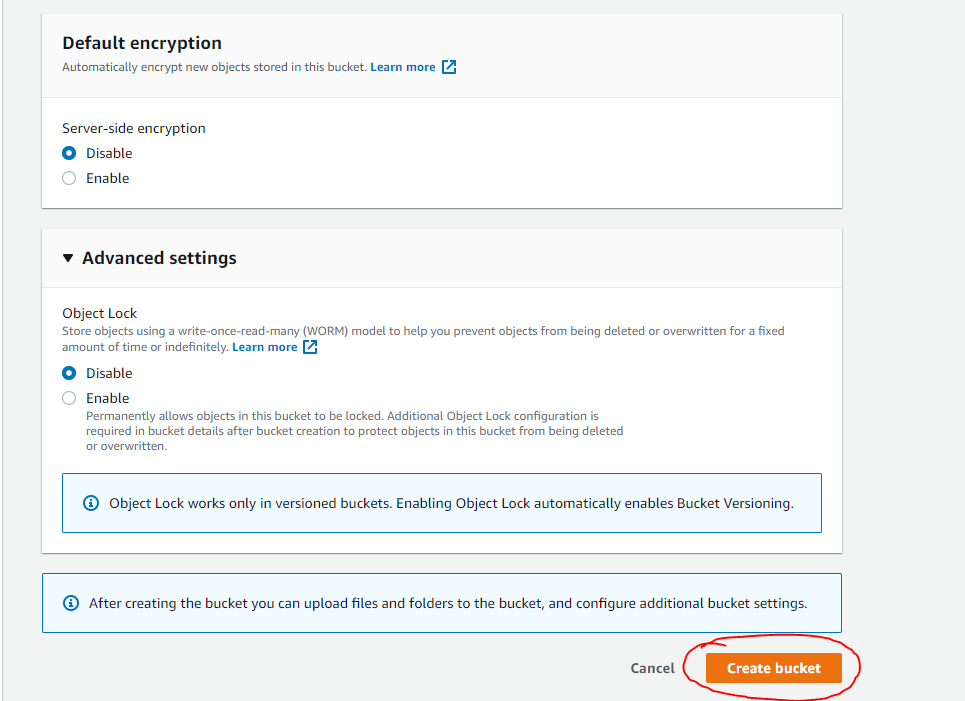
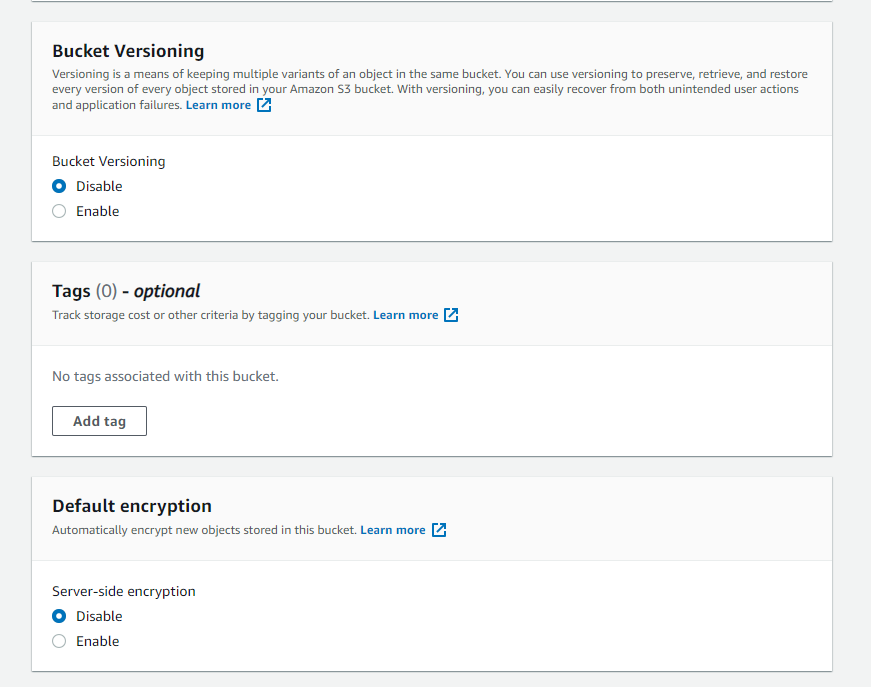
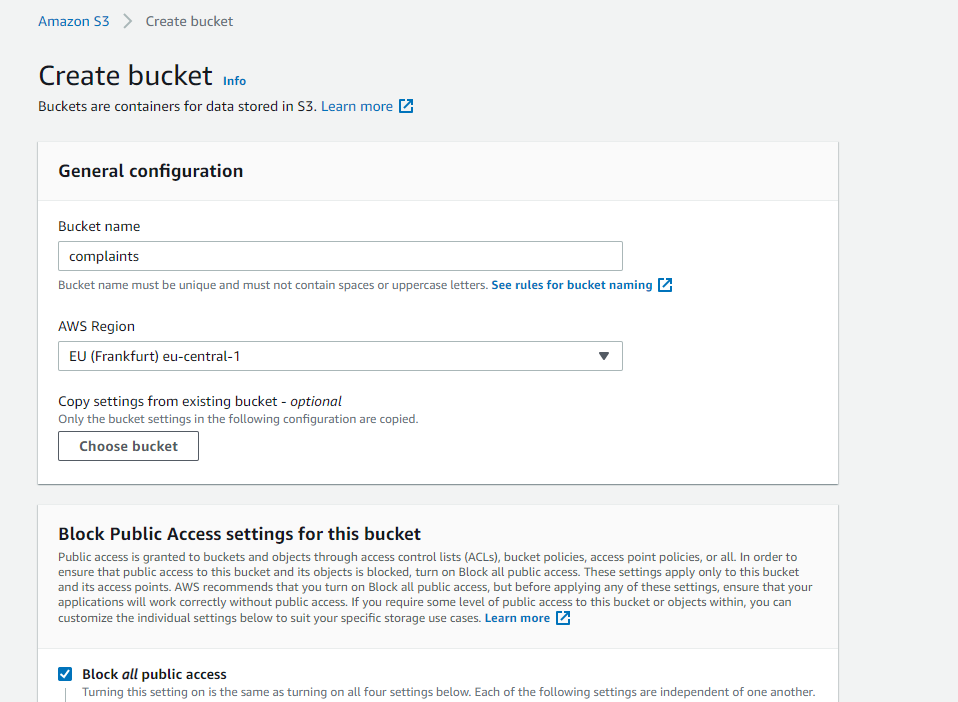
Once you have your AWS account ready and you are in the Console, you have to select **Services** and then **S3** under **Storage**



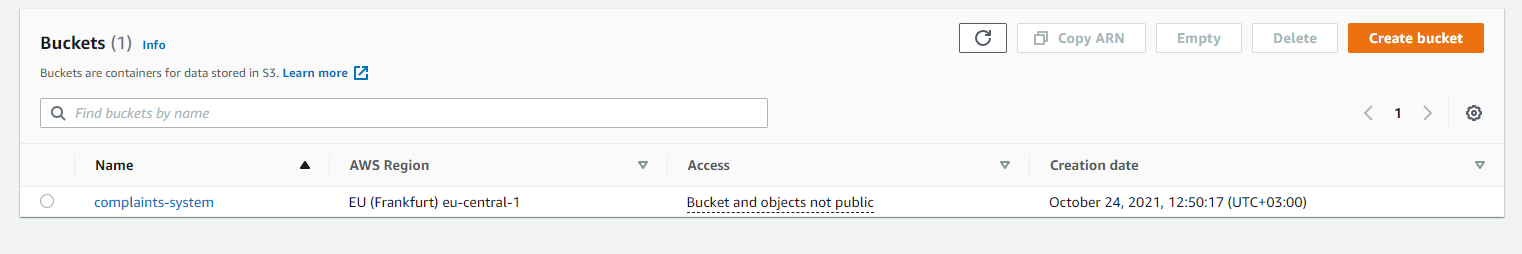
Now we need to create a bucket where we want to store our photos. Choose an appropriate name for the application and a close region to you. It will have some predefined security options. Leave them as they are for now. Later we will decide the read rights on the bucket:

Please uncheck “block public access option”.





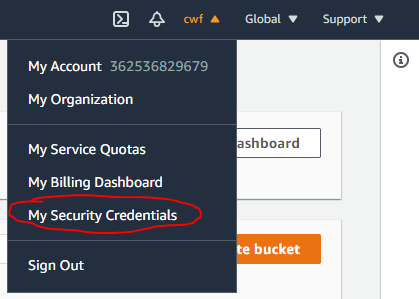
You can choose different name. At the end it will show you your existing buckets:

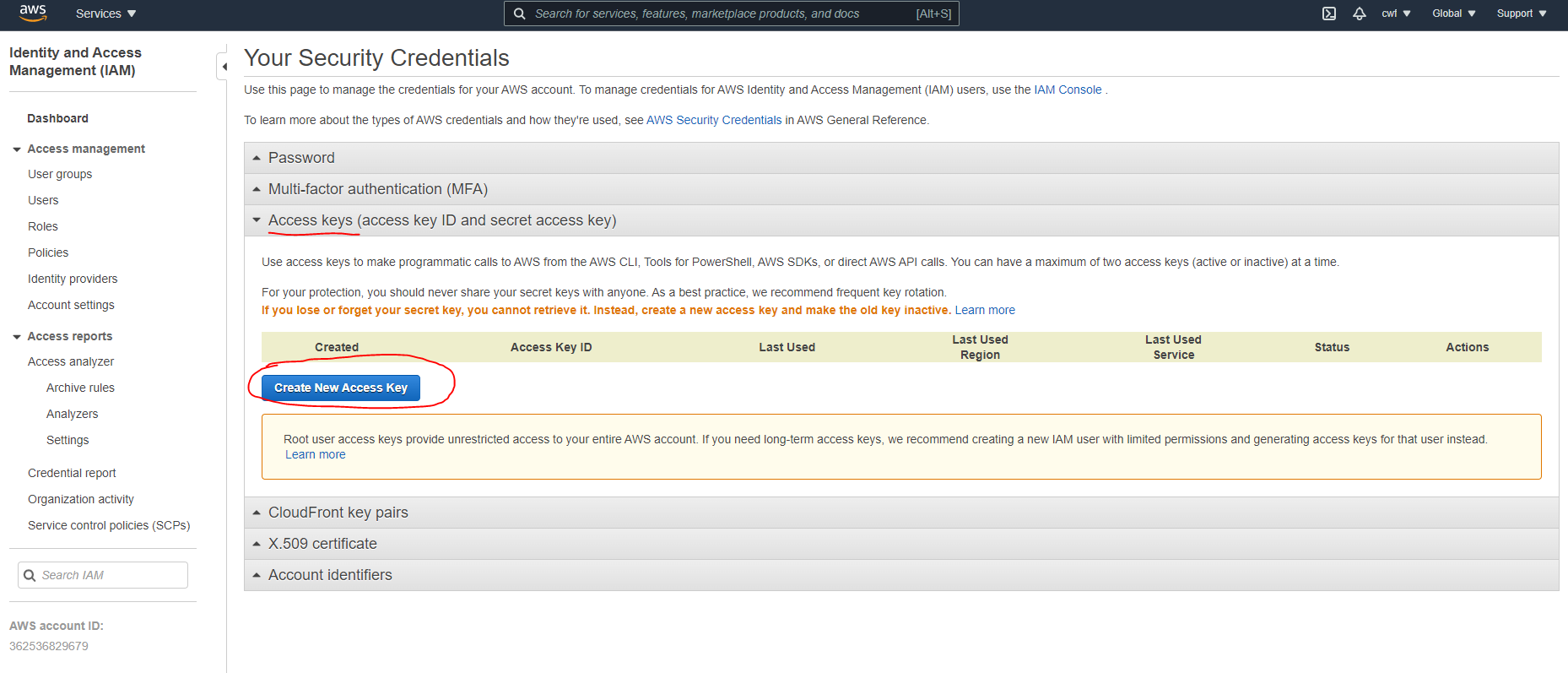


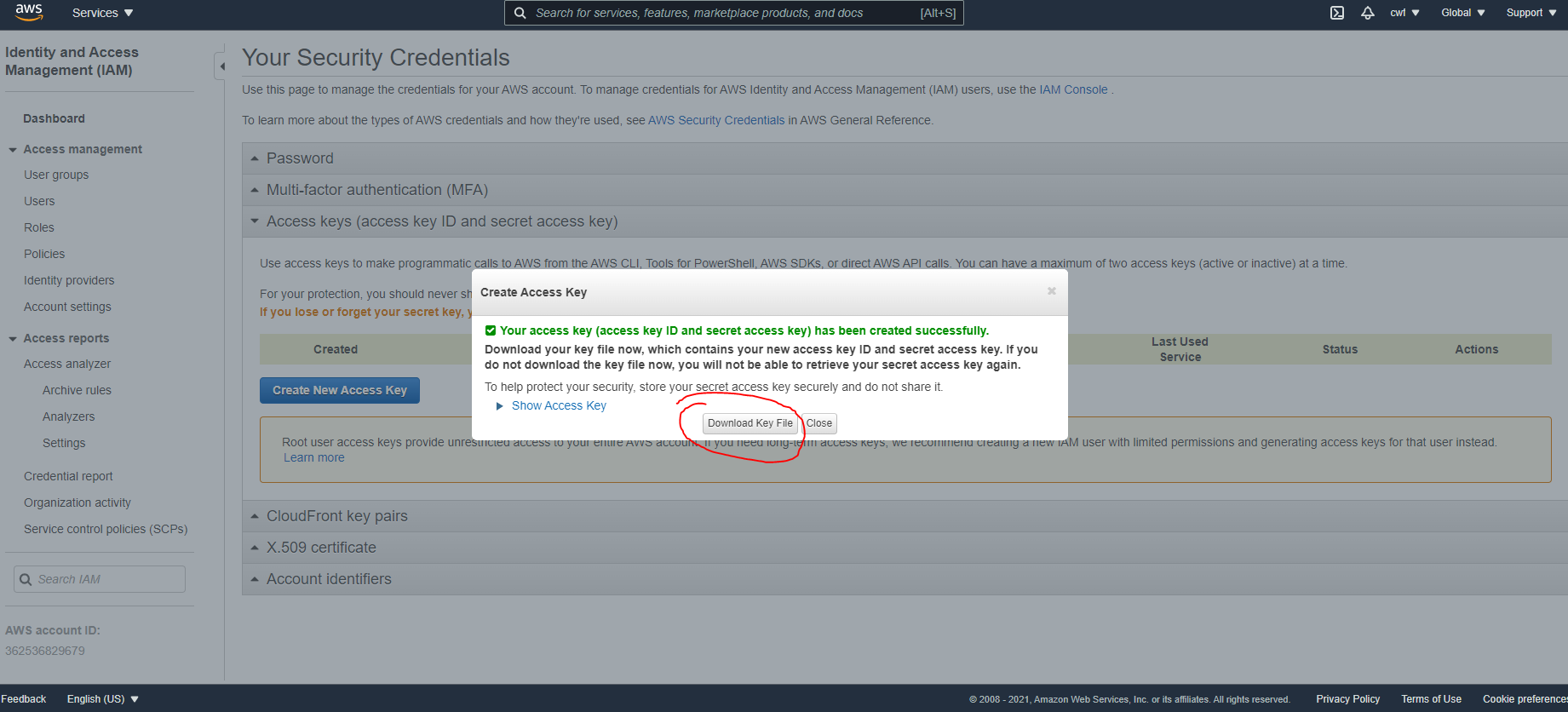
## Fetch credentials and integrate S3 in existing project

To connect to AWS services with boto3, you will need to generate a key and a secret. It is essential to store these credentials right after they are generated!

Go to the upright corner on the navigation bar and click your profile name, then follow the steps:







After you downloaded your credentials, you need to go to your **.env** file and define them:

AWS\_ACCESS\_KEY="PASTE YOUR KEY"  
AWS\_SECRET="PAASTE YOUR SECRET"

AWS\_BUCKET="PASTE YOUR BUCKET NAME"

AWS\_REGION="eu-central-1" # if you have chosen different region, the value will be different

We need to make some adjustments to the current schemas. In schemas/bases.py we need to change BaseComplainSchema

like this:

We will remove the photo\_url, because on input we will receive a base64 encoded string and after we store it in s3 on response we will return photo\_url.

from marshmallow import Schema, fields  
  
  
class BaseComplainSchema(Schema):  
 title = fields.String(required=True)  
 description = fields.String(required=True)  
 amount = fields.Float(required=True)

Now in request and response schemas the changes will be the following:

from marshmallow import fields  
from schemas.bases import BaseComplainSchema  
  
  
class RequestComplainSchema(BaseComplainSchema):  
 photo = fields.String(required=True)  
 photo\_extension = fields.String(required=True)

from marshmallow import fields  
from marshmallow\_enum import EnumField  
  
from models.enums import State  
from schemas.bases import BaseComplainSchema  
  
  
class ComplaintResponseSchema(BaseComplainSchema):  
 id = fields.Integer(required=True)  
 status = EnumField(State, by\_value=True)  
 create\_on = fields.DateTime(required=True)  
 photo\_url = fields.URL(required=True)

We will create a folder called 'temp\_files' under the project's root. Its purpose will be to store the files the user has sent after we decode them. Add it to .gitignore.

Next, in the root of our project we will create a file called **constants.py**

It is really important to work with join especially if you are on windows (because of the \\ and / difference in the unix alike systems and windows).

import os  
  
ROOT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
TEMP\_FILE\_FOLDER = os.path.join(ROOT\_DIR, **'temp\_files'**)

In **utils** folder create a file called **helpers.py.** Here we will define a function which will help us to decode the photo:

import base64  
  
from werkzeug.exceptions import BadRequest  
  
  
def decode\_photo(path, encoded\_string):  
 with open(path, **"wb"**) as f:  
 try:  
 f.write(base64.b64decode(encoded\_string.encode(**"utf-8"**)))  
 except Exception as ex:  
 raise BadRequest(**"Invalid photo encoding"**)

Now we will create our S3Service, responsible for the communication between our app and s3:

import boto3  
from botocore.exceptions import ClientError  
  
  
from decouple import config  
from werkzeug.exceptions import InternalServerError  
  
  
class S3Service:  
 def \_\_init\_\_(self):  
 self.key = config(**"AWS\_ACCESS\_KEY"**)  
 self.secret = config(**"AWS\_SECRET"**)  
 self.s3 = boto3.client(  
 **"s3"**, aws\_access\_key\_id=self.key, aws\_secret\_access\_key=self.secret,  
 )  
 self.bucket = config(**"AWS\_BUCKET"**)  
  
 def upload\_photo(self, path, key, ext):  
 try:  
 self.s3.upload\_file(path, self.bucket, key, ExtraArgs={**'ACL'**: **'public-read'**, **'ContentType'**: **f'image/**{ext}**'**})  
 return **f"https://**{config(**'AWS\_BUCKET'**)}**.s3.**{config(**'AWS\_REGION'**)}**.amazonaws.com/**{key}**"** except ClientError:  
 raise InternalServerError(**"S3 is not available at the moment"**)

In the init method we are setting up the key and the secret we obtained from the AWS console.

Then we define a function which uploads the file with the help of the s3 client from boto library and return the URL of the photo. The path is the temp\_folder/file\_name.extension which we have already decode and stored locally.

The key is the name and the extension of the file.

We have done so much, but now we have to update the manager to follow the newly requested functionality.

The create method of the ComplaintManager now will look like this:

import os

import uuid

from constants import TEMP\_FILE\_FOLDER

from db import db

from models.complaint import ComplaintModel

from models.enums import State

from services.s3 import S3Service

from utils.helpers import decode\_photo

**s3 = S3Service()**

….

@staticmethod

def create(data, complainer\_id):

"""

Decode the base64 encoded photo,

uploads it to s3 and set the photo url to

the s3 generated url.

Flushes the row

"""

data["complainer\_id"] = complainer\_id

**encoded\_photo = data.pop("photo")**

**extension = data.pop("photo\_extension")**

**name = f"{str(uuid.uuid4())}.{extension}"**

**path = os.path.join(TEMP\_FILE\_FOLDER, f"{name}")**

**decode\_photo(path, encoded\_photo)**

**url = s3.upload\_photo(path, name)**

**data["photo\_url"] = url**

c = ComplaintModel(\*\*data)

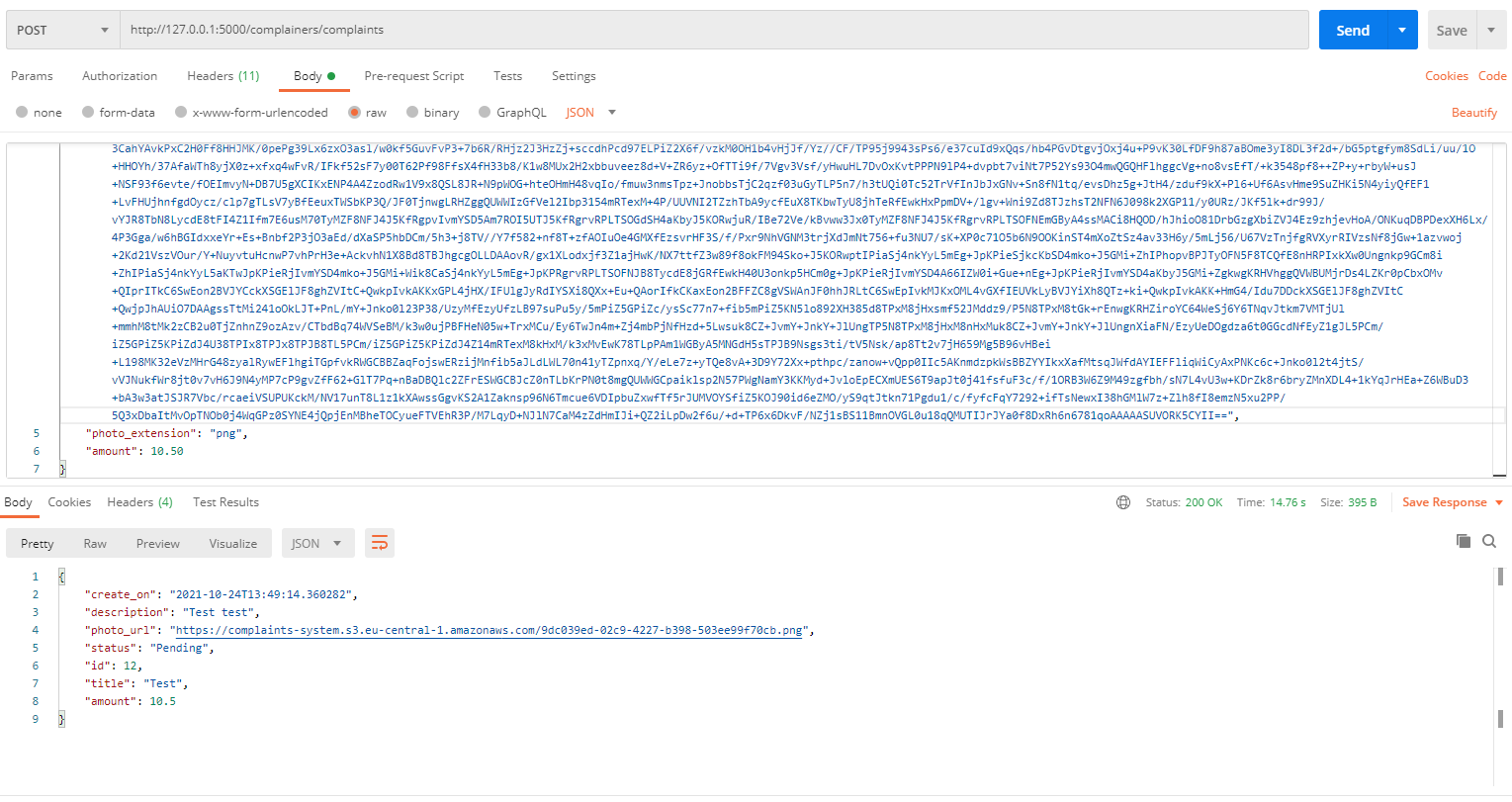
db.session.add(c)

db.session.flush()

return c

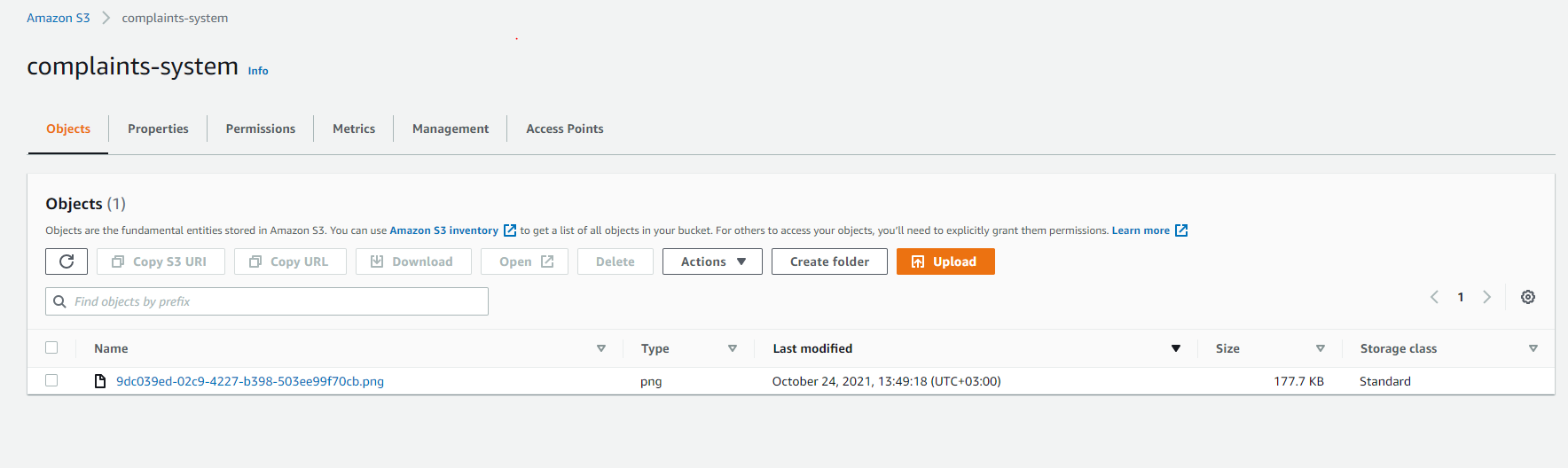
We added a couple of steps – we remove the **photo** and **photo\_extension** because they are not part of our model, but we store their values. We build a uuid and extension for the file name. Then we decode the photo (which will decode the file, give it this name and place it to the temp\_folder). Then using this path we will upload the photo to s3 with key the name we generated and save the URL to data and then to the database.

The request response will look like this:



You can try encode a string with this online [tool](https://www.base64-image.de/).  
All code changes in this section you can find in [this commit](https://github.com/InesIvanova/Flask-course-prep-materials/commit/5f5fb8d9d756693f78e76336753787112407a5e4).

You can validate the file is uploaded by checking the bucket’s content in the AWS Console.



## Approvers’ certificates

Your home assignment will be to make the schemas accept base64 encoded string and extension for the approver creation, convert the string to a photo, upload it to s3,store the URL in the database and return appropriate response schema. Reuse the code we have now for s3 service and decoding. If you face difficulties you can refer to [this commit](https://github.com/InesIvanova/Flask-course-prep-materials/commit/fd2031dc7d8ac1c5447c39090d6cefaec1971541).