1. Imagine um serviço de blog hosting "clássico" em que se paga um valor fixo por unidade de tempo (por exemplo x euros por mês), por exemplo, para se ter um blog disponível online, sujeito a valores máximos contratados para a quantidade de espaço em disco, tamanho da base de dados, e largura de banda.

➔ 1.1 Este serviço é uma situação de "Cloud Computing", pela definição NIST? Sim ou não, apenas. Não

➔ 1.2 Justifique a sua resposta anterior, elaborando sobre o que teria que ser diferente, para ter respondido o contrário.

A definição NIST refere cinco características essenciais para, genuinamente, se estar numa situação de Cloud Computing. Uma dessas características essenciais é a rápida elasticidade, entendida como a possibilidade de, dinamicamente, fazer (de)crescer os valores que definem o serviço, consoante a flutuação de necessidades. Assim, como os valores do caso apresentado têm tetos máximos, não podendo "rapidamente" escalar para cima, não é Computação na Nuvem estritamente de acordo com a definição NIST. Para que isso se verificasse, o utilizador deveria ter contratado um serviço dinâmico quanto ao espaço em disco, tamanho da base de dados e largura de banda.

2. Responda Verdadeiro (V) ou Falso (F), apenas:

➔ 2.1 AWS e GCP têm ambos um serviço "IAM" V

➔ 2.2 AWS e GCP têm ambos um serviço "S3" F

➔ 2.3 Um AWS root user é uma entidade equivalente a uma GCP Service Account F

➔ 2.4 Uma instância de um serviço AWS RDS, disponível nalguma AWS VPC, pode ser acedido diretamente na mesma VPC, por uma GCP compute instance. F

➔ 2.5 É possível fazer acesso SSH a uma compute instance AWS Cloud9, desde uma compute instance GCP V

➔ 2.6 É possível fazer prova de posse de nomes de domínio por TXT records, em AWS F

3.

Em cada uma das alíneas, escreva um único comando bash e/ou comando Git, assumindo uma instalação de Git em que o ramo por defeito é "master", para conseguir que:

➔ 3.1 é inicializado um Git repo vazio
git init

➔ 3.2 quando acontecerem Git commits, serão assinados pelo user com name "John Smith" e email "js@yes.org"
git config user.name "John Smith" && git config user.email "js@yes.org"

➔ 3.3 é criado um ficheiro de nome "1.txt" com conteúdo "1"
echo "1">1.txt

➔ 3.4 o ficheiro "1.txt" é adicionado ao stage
git add 1.txt

➔ 3.5 faz-se um commit com mensagem "c1"
git commit -m "c1"

➔ 3.6 altera-se o ficheiro "1.txt" adicionando-lhe uma última linha com o conteúdo "2"
echo "2">>1.txt

➔ 3.7 faz-se um novo commit, que já considera a nova versão de "1.txt", com mensagem "c2"
git commit --all -m "c2"

➔ 3.8 cria-se e muda-se do ramo "master", para um novo branch "alt"
git checkout -b alt

➔ 3.9 altera-se o ficheiro "1.txt" adicionando-lhe uma última linha com o conteúdo "alt final"
echo "alt final">>1.txt

➔ 3.10 faz-se um novo commit, que já considera a nova versão de "1.txt", com mensagem "c3@alt"
git commit --all -m "c3@alt"

➔ 3.11 muda-se do ramo "alt" para o ramo "master"
git checkout master

3.12 Assuma que escreveu perfeitos todos os comandos pedidos na questão anterior. A esses comandos, seguiu-se o comando
git checkout head~1 1.txt

➔ Qual é agora, na working directory, o conteúdo exato do ficheiro 1.txt?
1

4. Nesta questão deve escrever partes de um sistema Python Flask, a ser Cloud deployed.

A ideia é ter-se um frontend de procura sobre o histórico de conteúdos de TV e rádio, que https://www.rtp.pt/play , vai disponibilizando.

Para simplificar, assuma que cada conteúdo tem apenas o seu URL, uma data de primeira publicação, e um título.

As procuras são por título e incidem sobre toda a BD, mesmo quando os conteúdos já não estejam disponíveis para visualização no site da RTP.

A BD está online via um serviço RDS MySQL, escutando no porto 3306, mas o frontend só consulta um "dump" estático, em ficheiro de texto BD.JSON, que representa a mesma informação, e que acompanha o deployment da app.

Em backend, um outro código, que corre diariamente, identifica novos programas e acrescenta-os à BD MySQL.

Admita que o código de frontend é o ficheiro F.py ; admita que o código de backend é o ficheiro B.py .

4.1 Decida um formato JSON capaz de representar coleções de registos estruturados em <URL, title , date>. Apresente um exemplo concreto, para esclarecer a sua estrutura.

Assuma que a procura de programas só está disponível por "title". O valor chega a F.py por elemento com name "nameTitle".

➔ Escreva, com liberdade criativa, o Python Flask suficiente para

- servir [apenas] a route "/search" em F.py ,

- disponível por GET e por POST,

- que deve procurar em BD.JSON, respeitando o formato que decidiu, pelos registos cujo título contenha o valor recebido via nameTitle,

- e responder, também em JSON, os resultados.

Exemplo de estrutura JSON:

```
[
 [429, # id na BD MySQL
  "https://www.rtp.pt/play/p3285/e277322/andar-em-frente",  # URL
  "Em frente", # title
  "2023-2-23 17:2:47" # date
 ], ...
 [365,
 "https://www.rtp.pt/play/p8940/e554601/comecar-de-novo",
 "Começar de novo",
 "2023-57-23 16:57:57"
 ],  ...
]
```

```python
from flask import Flask, render_template, request
import json

app = Flask(__name__)

BD = "BD_SIMPLEST.JSON"

@app.route("/", methods=['GET', 'POST'])
def root():
    return render_template("search_interface.html")
# def root

@app.route("/search", methods=['GET', 'POST'])
def handle_search():
    search_exp = None
    if(request.method=='GET'):
        search_exp = request.args['nameTitle'].strip()
    elif(request.method=='POST'):
        search_exp = request.form['nameTitle'].strip()
    # if-else

  if(search!=None):
        findings = search(
            search_exp
        )

        jsonResults = json.dumps(findings)
        return jsonResults
    # if
  else:
        return "No search expression received, nothing done."
    # if
# def go_search

def search(pExp, pBD = BD):
    findings = []
    fr = open(pBD, 'r')
    bd:str = fr.read()
    oBD = json.loads(bd)

    for recordAsTuple in oBD:
        title:str = recordAsTuple[2]

        bMatch = title.find(pExp)!=-1
        if(bMatch):
            findings.append(recordAsTuple)
        # if
  # for
  return findings
# def search

if (__name__=='__main__'):
    app.run(
        debug=True
    )
```

4.2 Imagine que existe uma outra route "/updateDB" (que não tem que codificar) que atualiza o ficheiro BD.JSON, para refletir o MySQL.

➔ Sabendo que o frontend está deployed num endereço "E" e o backend em endereço "B", indique práticas de networking e de segurança que poderiam/deveriam ser seguidas neste sistema.

Quanto a networking, uma vez que F precisa de utilizar o serviço de MySQL em backend, esse serviço em B tem que estar configurado para aceitar "incoming MySQL traffic on port 3306".

Quanto a segurança, o serviço B pode restringir a regra de networking acima referida, para uma única origem, o endereço F. Pode também só aceitar tráfego encriptado por SSL, de clientes que se apresentem com os certificados adequados, tipicamente ficheiros .pem representativos da identidade do cliente (client-cert.pem) com a chave certa (client-key.pem) para acesso regulado por certa autoridade (server-ca.pem).