Sistemes Distribuïts

Lab1 - Introducció

Outline

1. Entorn de desenvolupament
2. Python
3. Model client-servidor
4. xmlrpc
5. grpc
6. http server
7. Redis

Entorn de desenvolupament

Entorn de desenvolupament

- Preferiblement Sistema operatiu basat amb UNIX:
  - Ubuntu, debian, MAC OS

- Windows amb Windows Subsystem for Linux (WSL) + imatge de Ubuntu 20.04 a través de la tenda de windows
  - https://docs.microsoft.com/es-es/windows/wsl/install-win10#manual-installation-steps

Entorn de desenvolupament

- Entorn de programació:
  - Preferiblement VSCode
  - https://code.visualstudio.com/download

- Python >= v3.6:
  - https://www.python.org/downloads/release/python-387/
  - $ python --version

Python

Python

- Python és un llenguatge de programació interpretat.
- Fàcil llegibilitat del seu codi.
- Es tracta d'un llenguatge de programació multiparadigma, ja que suporta parcialment l'orientació a objectes, programació imperativa i, en menor mesura, programació funcional.

Python

```
In [8]: # Creamos la clase Persona con los atributos nombre y edad

        class Persona:
            nombre = ''
            edad = ''

        # Creamos el objeto Pablo,30 años una instacia de la clase Persona

        pablo=Persona()
        pablo.nombre='Pablo'
        pablo.edad='30'
```

```
class UniversityStudent:

    def __init__(self,id,name,gender,university,career,numsubjects):
        self.id = id
        self.name = name
        self.gender = gender
        self.university = university
        self.career = career
        self.numsubjects = numsubjects

    def inscribeSubjects(self):
        pass

    def cancelSubjects(self):
        pass

    def consultRatings(self):
        pass
```

```
4   #
5   def suma (a, b, c):
6       resultado = a + b + c
7       print (resultado)
8
9
10  suma(1, 2, 3)
11
```

Python

- Python utilitza mòduls o llibreries
- El Python Package Index o PyPI és el repositori de programari oficial per a aplicacions de tercers en el llenguatge de programació Python.
- $ python3 -m pip --version
- $ python3 -m pip --list

```python
import requests

# Search GitHub's repositories for requests
response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
)
```

Model client servidor

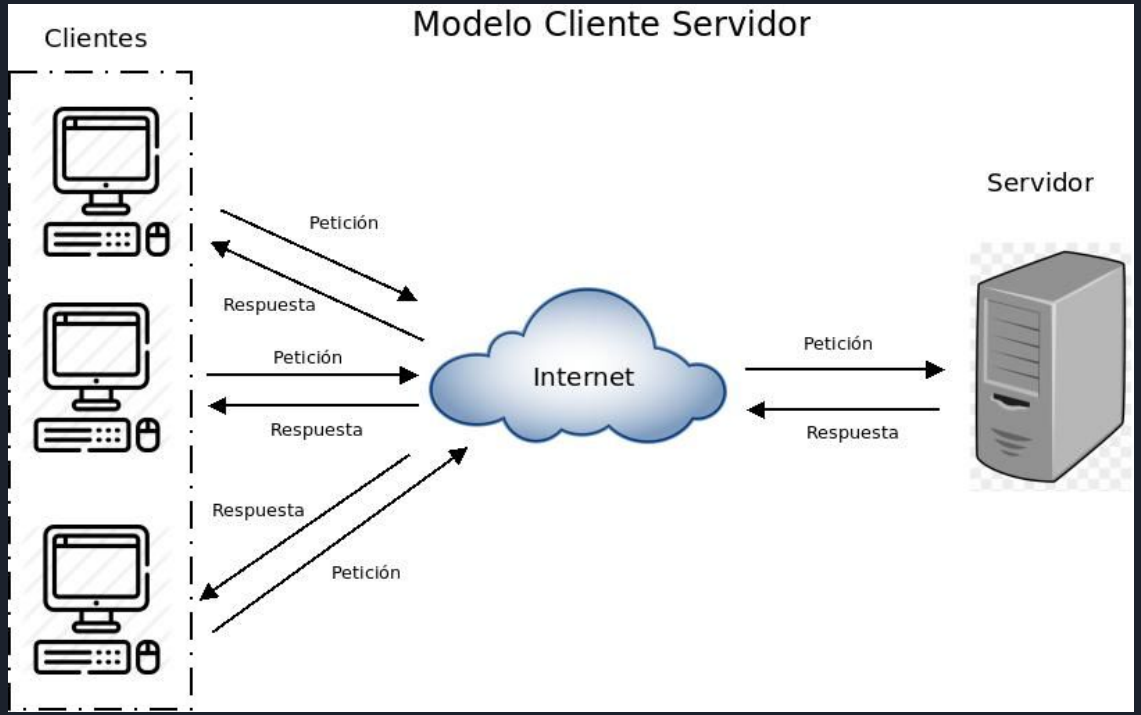
Model client servidor

- L'arquitectura client-servidor és un model de disseny de software en el qual les tasques es reparteixen entre els proveïdors de recursos, anomenats servidors, i els demandants, anomenats clients.
- Un client realitza peticions a un altre programa, el servidor, qui li dóna resposta.

Model client servidor

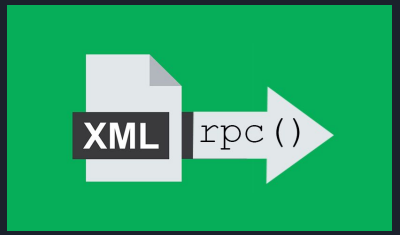
Examples:

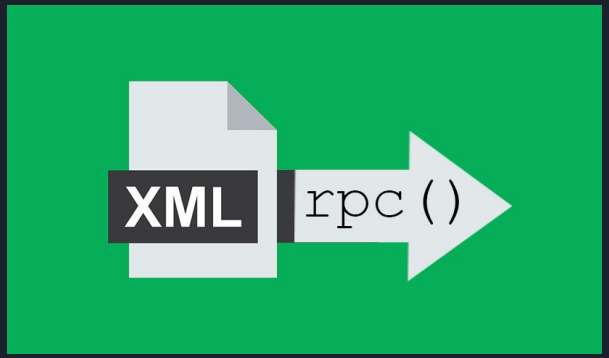
- Servidor web
- Servidor ftp
- Servidor DNS
- Servidor de correu
- etc, etc..
- Computació (Pràctica 1)



Model client servidor

- Comuincació RPC: Remote procedure call és una tecnologia que permet a un programa fer que una subrutina o procediment s'executi en un altre espai d'adreces (habitualment en un altre ordinador en una xarxa compartida).
  - xml-rpc
  - grpc




XML
rpc()

xml-rpc

- És un protocol de crida a procedimiento remot que utilitza XML per codificar les dades i HTTP com a protocol de transmissió de missatgeria.
- És un protocol molt simple ja que només defineix uns quants tipus de dades i comandos útils, a més d'una descripció completa de curta extensió.

xml-rpc

- Una invocación **XML-RPC** podría ser:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>org.wikipedia.intercambioDatos</methodName>
  <params>
    <param>
       <value><i4>360</i4></value>
    </param>
    <param>
       <value><i4>221</i4></value>
    </param>
  </params>
</methodCall>
```

- Una respuesta a la invocación:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Intercambio datos nro. 360 por 221</string></value>
    </param>
  </params>
</methodResponse>
```

xml-rpc

Servidor

xmlrpc_function.py

```python
from xmlrpc.server import SimpleXMLRPCServer
import logging
import os

# Set up logging
logging.basicConfig(level=logging.INFO)

server = SimpleXMLRPCServer(
    ('localhost', 9000),
    logRequests=True,
)


# Expose a function
def list_contents(dir_name):
    logging.info('list_contents(%s)', dir_name)
    return os.listdir(dir_name)


server.register_function(list_contents)

# Start the server
try:
    print('Use Control-C to exit')
    server.serve_forever()
except KeyboardInterrupt:
    print('Exiting')
```

Client

xmlrpc_function_client.py

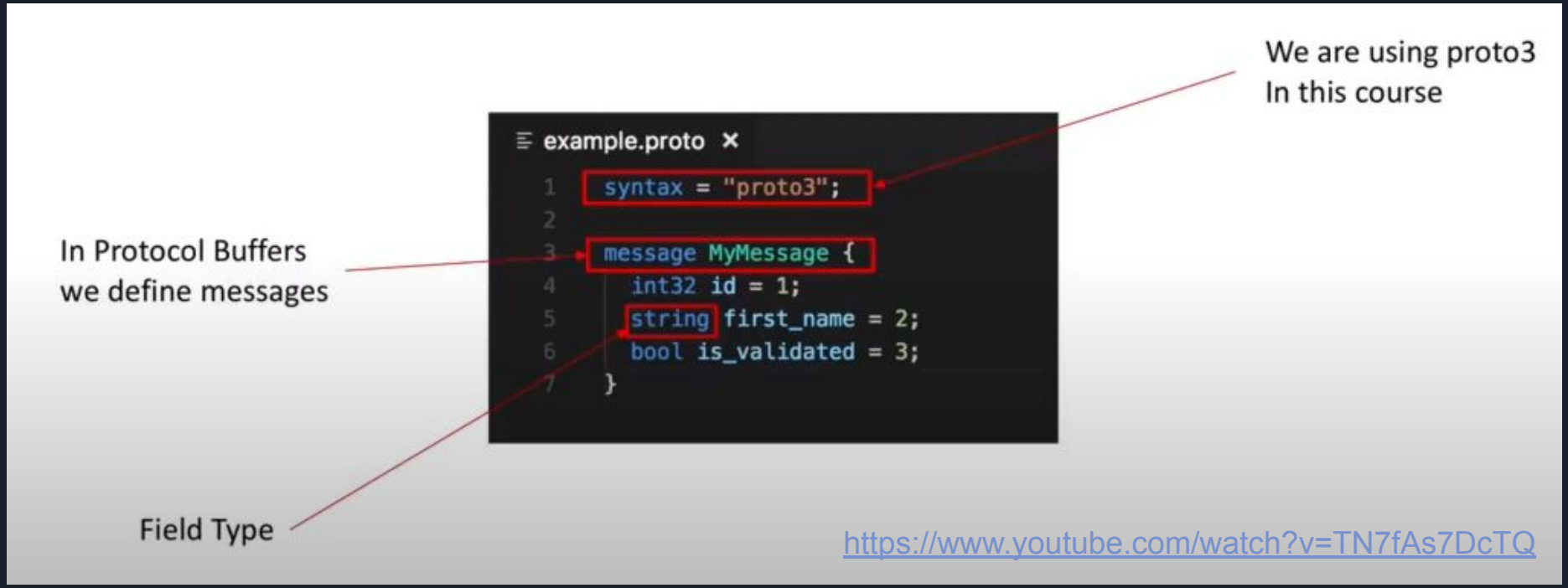
```python
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy('http://localhost:9000')
print(proxy.list_contents('/tmp'))
```


GRPC

grpc

- Eun un protocol de crida de procediment remot que utilitza com transport **HTTP/2**, i **Protocol Buffers** com a llenguatge de descripció d'interfície.
- Protocol buffers es un mecanisme extensible, neutral pel que fa a el llenguatge i la plataforma, per serialitzar dades estructurades; pensi en XML, però més petit, més ràpid i més simple.

grpc


We are using proto3
In this course
example.proto
1 syntax = "proto3";
2
3 message MyMessage {
4 int32 id = 1;
5 string first_name = 2;
6 bool is_validated = 3;
7 }
In Protocol Buffers
we define messages
Field Type
https://www.youtube.com/watch?v=TN7fAs7DcTQ

grpc

1. Creem la funció

```python
import math

def square_root(x):
  y = math.sqrt(x)
  return y
```

calculator.py hosted with ❤ by GitHubview raw

2. Definim el fitxer .proto

```
syntax = "proto3";

message Number {
    float value = 1;
}

service Calculator {
    rpc SquareRoot(Number) returns (Number) {}
}
```

calculator.proto hosted with ❤ by GitHub view raw

3. Generem les classes gRPC per a python

```
$ pip install grpcio
$ pip install grpcio-tools

$ python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. calculator.proto
```

calculator_pb2.py

calculator_pb2_grpc.py

grpc

- calculator_pb2.py — Conté les classes dels missatges
  - calculator_pb2.Number for request/response variables (x and y)

- calculator_pb2_grpc.py — Conté les classes client i servidor
  - calculator_pb2_grpc.CalculatorServicer **per al servidor**
  - calculator_pb2_grpc.CalculatorStub **per al client**

grpc

Servidor

```python
import grpc
from concurrent import futures
import time

# import the generated classes
import calculator_pb2
import calculator_pb2_grpc

# import the original calculator.py
import calculator

# create a class to define the server functions, derived from
# calculator_pb2_grpc.CalculatorServicer
class CalculatorServicer(calculator_pb2_grpc.CalculatorServicer):

    # calculator.square_root is exposed here
    # the request and response are of the data type
    # calculator_pb2.Number
    def SquareRoot(self, request, context):
        response = calculator_pb2.Number()
        response.value = calculator.square_root(request.value)
        return response
```

```python
# create a gRPC server
server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))

# use the generated function `add_CalculatorServicer_to_server`
# to add the defined class to the server
calculator_pb2_grpc.add_CalculatorServicer_to_server(
        CalculatorServicer(), server)

# listen on port 50051
print('Starting server. Listening on port 50051.')
server.add_insecure_port('[::]:50051')
server.start()

# since server.start() will not block,
# a sleep-loop is added to keep alive
try:
    while True:
        time.sleep(86400)
except KeyboardInterrupt:
    server.stop(0)
```

server.py hosted with ❤ by GitHub view raw

grpc

Estructura del projecte

```
basic-grpc-python/
├── calculator.py          # module containing a function
|
├── calculator.proto       # protobuf definition file
|
├── calculator_pb2_grpc.py # generated class for server/client
├── calculator_pb2.py      # generated class for message
|
├── server.py              # a server to expose the function
└── client.py              # a sample client
```

https://www.semantics3.com/blog/a-simplified-guide-to-grpc-in-python-6c4e25f0c506/

https://grpc.io/docs/languages/python/quickstart/

Client

```python
import grpc

# import the generated classes
import calculator_pb2
import calculator_pb2_grpc

# open a gRPC channel
channel = grpc.insecure_channel('localhost:50051')

# create a stub (client)
stub = calculator_pb2_grpc.CalculatorStub(channel)

# create a valid request message
number = calculator_pb2.Number(value=16)

# make the call
response = stub.SquareRoot(number)

# et voilà
print(response.value)
```

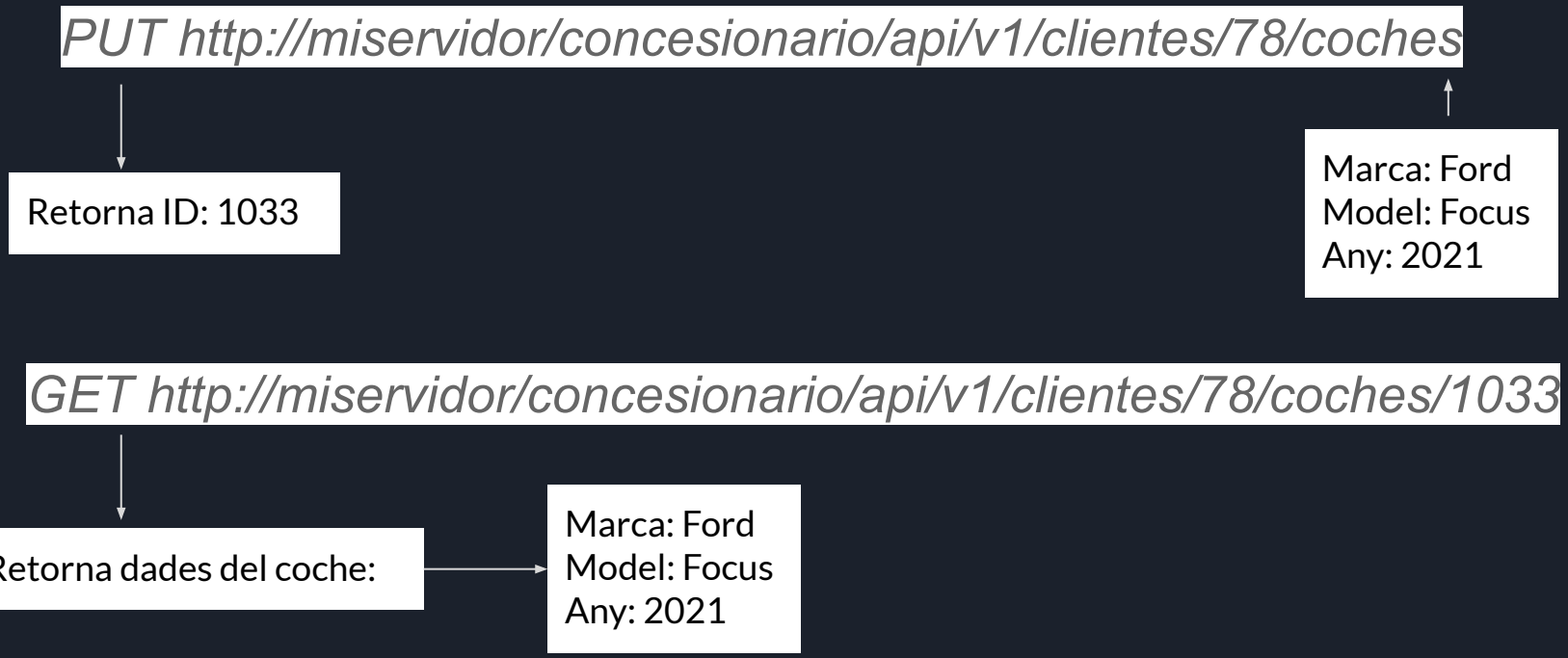
client.py hosted with ❤ by GitHub view raw

http server

Http server

- Un servidor http proprciona una interficie web per accedir a serveis remots a través de una API REST.
- API REST: Protocol client / servidor sense estat: cada petició HTTP (url) conté tota la informació necessària per a executar-la.
- Les operacions més importants relacionades amb les dades en qualsevol sistema REST i l'especificació HTTP són quatre: POST (crear), GET (llegir i consultar), PUT (editar) i DELETE (eliminar).

Http server


PUT http://miservidor/concesionario/api/v1/clientes/78/coches
Retorna ID: 1033
Marca: Ford
Model: Focus
Any: 2021
GET http://miservidor/concesionario/api/v1/clientes/78/coches/1033
Retorna dades del coche:
Marca: Ford
Model: Focus
Any: 2021

Http server

- Llibreria python: Flask

Python

```python
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello():
    return "Hello World!"

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

http://127.0.0.1/

Python

```python
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello():
    return "Hello World!"


@app.route('/<name>')
def hello_name(name):
    return "Hello {}!".format(name)

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

http://127.0.0.1/jordi

https://flask.palletsprojects.com/en/1.1.x/

redis

Redis

- Redis és un magatzem d'estructures de dades en memòria de codi obert (amb llicència BSD)
- S'utilitza com a base de dades, memòria cache i message broker (cues).
- Redis proporciona estructures de dades com ara strings, hash, llistes, sets, sets ordenats amb consultes d'interval, mapes de bits, hiperglògens, índexs geoespacials i streams.

Redis

- Instal·lació servidor:

  `MAC OS: brew install redis`

  `Ubuntu/debian: sudo apt-get install redis-server`

- Instal·lació client: `python3 -m pip install -U redis`

Python >>>

```
>>> import redis
>>> r = redis.Redis()
>>> r.mset({"Croatia": "Zagreb", "Bahamas": "Nassau"})
True
>>> r.get("Bahamas")
b'Nassau'
```

https://redis-py.readthedocs.io/en/stable/