

Práctica 1, Parte 2. Haciendo una calculadora con Apache Thrift

0. Introducción

Para esta práctica se trata de hacer una calculadora donde a través de un cliente que envía una petición a un servidor para realizar varios cálculos y devolver el resultado al cliente, quien lo imprime en pantalla.

En principio se trataba de hacer las operaciones básicas (sumas, restas, productos y cocientes), al cual poco a poco se ha ido añadiendo más complejidad haciendo operaciones acumulativas, con structs, con vectores y con matrices.

Además, como variación con respecto a la parte 1 de esta práctica, se ha tratado de implementar el cliente en un lenguaje (Python en este caso) y el servidor en otro (se intentó con Dart primero, pero por cuestiones que detallaré más tarde decidí hacerlo en Ruby).

1. Una calculadora básica.

Lo primero que se hizo fue tratar de hacer que la calculadora pueda sumar, restar, multiplicar y dividir. Para ello se necesitan dos números y la operación a realizar, junto con el host del servidor, ya que sin el servidor el cliente no funcionaría adecuadamente (cogería las operaciones pero no haría nada más).

Por ello, en el archivo de definiciones se establecieron 4 funciones:

```
void ping(),
i32 suma(1:i32 num1, 2:i32 num2),
i32 resta(1:i32 num1, 2:i32 num2),
i32 producto(1:i32 num1, 2:i32 num2),
double cociente(1:i32 num1, 2:i32 num2),
```

Al generar las plantillas, en el servidor se crean las funciones arriba mencionadas, con un atributo de salida return que recomiendan rellenar.

Para las operaciones básicas sólo había que relacionar ambos operandos con la operación de la función. Así, en el caso de la suma tendríamos :

Ruby	Python	Dart
<pre>def suma(n1,n2) puts "Sumando" return n1+n2 end</pre>	<pre>def suma(self, n1, n2): print("sumando ") return n1 + n2</pre>	<pre>@override Future<int> suma(int num1, int num2) async { print("Suma"); return num1 + num2; }</pre>

En el cliente se usa la función `switch_basico` el cual observa la operación a realizar de entre las cuatro posibles (+, -, x, :) y según ese operando llama a una u otra función pasando como parámetros los operandos y el operador.

```
def switch_basico(operacion, oper1, oper2):
    if(operacion == '+'):
        resultado = client.suma(oper1, oper2)
    elif(operacion == '-'):
        resultado = client.resta(oper1, oper2)
    elif(operacion == 'x'):
        resultado = client.producto(oper1, oper2)
    elif(operacion == ':'):
        resultado = client.cociente(oper1, oper2)
    else:
        resultado = None
    return resultado
```

The image shows two terminal windows. The top window, titled 'Servidor', shows the server running and performing operations: 'me han hecho ping()', 'sumando 32 con 64', 'restando 32 con 64', 'Haciendo producto de 32 y 64', and 'Haciendo cociente de 32 y 64'. The bottom window, titled 'Cliente', shows the client running and sending requests to the server: 'hacemos ping al server', '32 + 64 = 96', 'hacemos ping al server', '32 - 64 = -32', 'hacemos ping al server', '32 x 64 = 2048', 'hacemos ping al server', and '32 : 64 = 0.5'.

1.1 Intentando hacer el servidor en otro lenguaje

Llegados a este punto, se barajó la posibilidad de hacer el programa del servidor en otro lenguaje. Por tanto, nada más ver que funcionaban las operaciones básicas en un servidor escrito con Python, me aventuré a intentar hacerlo con Dart.

Para ello, me basé en el ejemplo que ofrecía Apache Thrift sobre ese lenguaje, pero al adaptarlo para que se ajustara a la práctica, observé que sólo podía hacer una operación antes de salirse al Terminal o Bash. En un principio pensé en que simplemente se hiciera un bucle infinito ejecutando el programa, de forma de que si acabara no hubiera que ejecutarlo manualmente, pero pensé : *¿Y si mientras se rearma el servidor llega una petición? No lo podría atender y daría error en el cliente.*

The image shows two terminal windows. The top window has two tabs: 'Cliente' and 'Servidor'. The 'Servidor' tab is active, showing the command `dart servidor.dart` and its output: `Warning: Interpreting this as package URI, 'package:calculadora/servidor.dart'.`, `listening for TCP connections on 9090`, `connected`, `ping()`, and `Suma`. The 'Cliente' tab shows the command `dart servidor.dart` and its output: `Warning: Interpreting this as package URI, 'package:calculadora/servidor.dart'.`, `listening for TCP connections on 9090`, `connected`, `ping()`, and `Cociente`. The bottom window also has two tabs: 'Cliente' and 'Servidor'. The 'Servidor' tab is active, showing the command `python3 cliente.py 32 + 64` and its output: `hacemos ping al server`, `32 + 64 = 96`, `hacemos ping al server`, `32 : 64 = 0,5`, and `ivan@Arcadia-TUF: /mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$`. The 'Cliente' tab shows the command `python3 cliente.py 32 + 64` and its output: `hacemos ping al server`, `32 + 64 = 96`, `hacemos ping al server`, `32 : 64 = 0,5`, and `ivan@Arcadia-TUF: /mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$`.

También fue interesante fijarse en que, si bien lenguajes como Python, Ruby o Java tenían una clase denominada **TSimpleServer**, Dart no lo tenía. Así pues, descarté avanzar con el desarrollo del servidor en Dart (aunque se quedará adjunto en la entrega) y hacerlo en Ruby.

Dado que en Ruby sí permitía seguir la ejecución del servidor tras servir una petición del cliente, seguí con él.

The image shows two terminal windows. The top window has two tabs: 'Cliente' and 'Servidor'. The 'Servidor' tab is active, showing the command `ruby servidor.rb` and its output: `Inicio servidor...`, `ping()`, `Sumando`, `ping()`, `Restando`, `ping()`, `Multipliando`, `ping()`, and `Dividiendo`. The 'Cliente' tab shows the command `ruby servidor.rb` and its output: `Inicio servidor...`, `ping()`, `Sumando`, `ping()`, `Restando`, `ping()`, `Multipliando`, `ping()`, and `Dividiendo`. The bottom window also has two tabs: 'Cliente' and 'Servidor'. The 'Servidor' tab is active, showing the command `python3 cliente.py 32 + 64` and its output: `hacemos ping al server`, `32 + 64 = 96`, `hacemos ping al server`, `32 - 64 = -32`, `hacemos ping al server`, `32 x 64 = 2048`, `hacemos ping al server`, `32 : 64 = 0,5`, and `ivan@Arcadia-TUF: /mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$`. The 'Cliente' tab shows the command `python3 cliente.py 32 + 64` and its output: `hacemos ping al server`, `32 + 64 = 96`, `hacemos ping al server`, `32 - 64 = -32`, `hacemos ping al server`, `32 x 64 = 2048`, `hacemos ping al server`, `32 : 64 = 0,5`, and `ivan@Arcadia-TUF: /mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$`.

2. Introduciendo una función con *structs*: Resolver ecuaciones cuadráticas

En una ecuación cuadrática hay dos soluciones, cosa que no se podría hacer con una función ya que no puede devolver dos enteros de por sí. Pero... ¿y si los metemos en un *struct* junto con los valores que se pasan, para comprobar que lo que se recibe es el resultado de lo que se envía?

Así, lo primero que hay que hacer es declarar el struct en **calculadora.thrift** y una función que devuelva uno de ellos:

```
struct funcion_cuadratica{
  1: required i32 a
  2: required i32 b
  3: required i32 c
  4: required double x1
  5: required double x2
}
/* ... */

funcion_cuadratica ecuacionGradoDos(1: i32 a, 2: i32 b, 3: i32 c),
```

Luego, en el servidor se resuelve usando la fórmula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, que en el caso de que $a = 0$, $x = \frac{-c}{b}$.

```
def ecuacionGradoDos(a,b,c)
  puts "ECUACION"
  result = Funcion_cuadratica.new
  result.a = a
  result.b = b
  result.c = c

  if (result.a == 0)
    result.x1 = (-result.c * 1.0)/(result.b * 1.0)
    result.x2 = (-result.c * 1.0)/(result.b * 1.0)
  else
    discriminante = (result.b*result.b) - 4*result.a*result.c
    if (discriminante < 0)
      result.x1 = -12345678
      result.x2 = -12345678
    else
      raiz = Math.sqrt(discriminante)
```

```

    result.x1 = (((result.b * result.b) + raiz)*1.0)/(2.0*result.a)
    result.x2 = (((result.b * result.b) - raiz)*1.0)/(2.0*result.a)
end
end
puts "Resultado extraido"
return result
end

```

Y en el cliente se usa el modificador “-s”, cambiando así el input de los argumentos al formato `python3 cliente.py -s <Ax^2> <Bx> <C>`.

The image shows two terminal windows. The left window, titled 'Cliente', shows the output of the server script 'ruby servidor.rb'. It prints 'Iniciando servidor...', 'ping()', 'ECUACION', 'Resultado extraido', and then a series of 'ping()' and 'ECUACION' messages. The right window, titled 'Servidor', shows the output of the client script 'python3 cliente.py' with various arguments. It prints 'hacemos ping al server', 'Para la ecuación 3x^2 + -5x + 1 -> X1 = 4.767591879243998 | X2 = 3.5657414540893355', and then a series of 'hacemos ping al server' and 'Para la ecuación' messages.

3. Vectores : Operaciones acumuladas

En el caso de los vectores, habría que definirlo en `calculadora.thrift` usando `list<TIPO>`. Así pues, definimos un array de enteros y algunas operaciones para las que usar un solo vector.

```

double sumaAcumulada(1: list<i32> acumulado),
double restaAcumulada(1: list<i32> acumulado),
double productoAcumulado(1: list<i32> acumulado),

```

Así pues, **secuencia** se encuentra como un *struct* con un tamaño y un puntero al *array*.

En el servidor, se coge el *array*, sacamos la primera variable como referencia y se va sumando los demás, devolviendo la operación acumulada en **result**.

```
def sumaAcumulada(arrayAcumulativo)
  acumulado = 0
  for item in arrayAcumulativo do
    acumulado = acumulado + item
  end
  return acumulado
end
```

En el cliente, además de poner otro modificador para aceptar leer los archivos, se han tenido que hacer funciones para leer desde archivo los valores del *array*.

El formato sería **python3 <Programa> -v1 <Archivo_vector> <Operacion>**

```
with open(argv[2]) as f:
    vector = [int(x) for x in f.read().split()]

def switch_acumulativo(operacion, acumulativo):
    if(operacion == '+'):
        resultado = client.sumaAcumulada(acumulativo)
    elif(operacion == '-'):
        resultado = client.restaAcumulada(acumulativo)
    elif(operacion == 'x'):
        resultado = client.productoAcumulado(acumulativo)
    else:
        resultado = None
    return resultado
```

```

ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-rb$ ruby servidor.rb
Iniciando servidor...
ping()
ping()
ping()
ping()
ping()
ping()
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or1.txt +
hacemos ping al server
[+] [30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 77
4, 694, 682, 29, 463, 582, 392, 369] = 15527.0
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or1.txt -
hacemos ping al server
[-] [30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 77
4, 694, 682, 29, 463, 582, 392, 369] = -15467.0
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or1.txt x
hacemos ping al server
[x] [30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 77
4, 694, 682, 29, 463, 582, 392, 369] = 4.405350181655338e+79
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or2.txt +
hacemos ping al server
[+] [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70, 26, 2, 90, 74, 77, 44, 95, 29
, 36] = 1640.0
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or2.txt -
hacemos ping al server
[-] [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70, 26, 2, 90, 74, 77, 44, 95, 29
, 36] = -1580.0
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v1 ../input_files/vect
or2.txt x
hacemos ping al server
[x] [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70, 26, 2, 90, 74, 77, 44, 95, 29
, 36] = 3.905205006545032e+49

```

4. Vectores : Operaciones entre vectores

Siguiendo con los vectores, ahora trataremos de hacer operaciones con dos vectores y devolverla en otro.

En el calculadora.thrift:

```

secuencia sumaVectores(secuencia,secuencia) = 9;
secuencia restaVectores(secuencia,secuencia) = 10;
secuencia productoVectores(secuencia,secuencia) = 11;
secuencia cocienteVectores(secuencia,secuencia) = 12;

```

En el servidor, cuando recibimos ambos arrays, reservamos memoria para otro array en base a la longitud de los dos argumentos.

```

def sumaVectores(v1, v2)
  total = Array.new(v1.length)
  for i in 0..v1.length-1 do
    total[i] = v1[i] + v2[i]
  end
  return total
end

```

En el cliente hacemos las comprobaciones de longitud, para ver que una no sea mayor o menor que la otra. En caso contrario, avisará y terminará la ejecución. Su ejecución sería **python3 cliente.py -v2 <archivo1> <operación> <archivo2>**

Para usar la función , la estructura sería

```
/* ... */
elif (argc == 5 and argv[1] == '-v2'):
    operacion = argv[3]
    ficheroAcumulado1 = argv[2]
    ficheroAcumulado2 = argv[4]

    with open(argv[2]) as f:
        vector1 = [int(x) for x in f.read().split()]

    with open(argv[4]) as f2:
        vector2 = [int(x) for x in f2.read().split()]

    resultado = switch_vectorial(operacion,vector1,vector2)

    if (resultado != None):
        print(f"{vector1} {operacion} {vector2} = {resultado}")
    else:
        uso()

/* ... */
```



```

ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-rb$ ruby servidor.rb
Iniciando servidor...
ping()
ping()
ping()
ping()
ping()
ori.txt + ../input_files/vector2.txt
hacemos ping al server
[30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 774, 6
94, 682, 29, 463, 582, 392, 369] + [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70
, 26, 2, 90, 74, 77, 44, 95, 29, 36] = [60.0, 442.0, 983.0, 356.0, 301.0, 629.0, 776.0, 1014.0, 947.0, 521.0, 354.0, 108
4.0, 595.0, 122.0, 827.0, 455.0, 644.0, 948.0, 136.0, 712.0, 236.0, 429.0, 164.0, 776.0, 784.0, 756.0, 106.0, 507.0, 677
.0, 421.0, 405.0]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v2 ../input_files/vect
ori.txt - ../input_files/vector2.txt
hacemos ping al server
[30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 774, 6
94, 682, 29, 463, 582, 392, 369] - [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70
, 26, 2, 90, 74, 77, 44, 95, 29, 36] = [0.0, 434.0, 871.0, 296.0, 133.0, 437.0, 660.0, 850.0, 913.0, 425.0, 318.0, 906.0
, 485.0, -24.0, 771.0, 311.0, 552.0, 934.0, 92.0, 524.0, 60.0, 289.0, 112.0, 772.0, 604.0, 608.0, -48.0, 419.0, 487.0, 3
63.0, 333.0]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v2 ../input_files/vect
ori.txt x ../input_files/vector2.txt
hacemos ping al server
[30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 774, 6
94, 682, 29, 463, 582, 392, 369] x [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70
, 26, 2, 90, 74, 77, 44, 95, 29, 36] = [900.0, 1752.0, 5192.0, 9780.0, 18228.0, 51168.0, 41644.0, 76424.0, 15810.0, 227
04.0, 6048.0, 88555.0, 29700.0, 3577.0, 22372.0, 27576.0, 27508.0, 6587.0, 2508.0, 58092.0, 13024.0, 25130.0, 3588.0, 15
48.0, 62460.0, 50468.0, 2233.0, 20372.0, 55290.0, 11368.0, 13284.0]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -v2 ../input_files/vect
ori.txt : ../input_files/vector2.txt
hacemos ping al server
[30, 438, 927, 326, 217, 533, 718, 932, 930, 473, 336, 995, 540, 49, 799, 383, 598, 941, 114, 618, 148, 359, 138, 774, 6
94, 682, 29, 463, 582, 392, 369] : [30, 4, 56, 30, 84, 96, 58, 82, 17, 48, 18, 89, 55, 73, 28, 72, 46, 7, 22, 94, 88, 70
, 26, 2, 90, 74, 77, 44, 95, 29, 36] = [1.0, 109.5, 16.553571428571427, 10.866666666666667, 2.5833333333333335, 5.552083
333333333, 12.379310344827585, 11.365853658536585, 54.705882352941174, 9.854166666666666, 18.666666666666668, 11.1797752
80898877, 9.818181818181818, 0.6712328767123288, 28.535714285714285, 5.319444444444445, 13.0, 134.42857142857142, 5.1818
181818182, 6.574468085106383, 1.6818181818181819, 5.128571428571429, 5.3076923076923075, 387.0, 7.711111111111111, 9.2
16216216216216, 0.37662337662337664, 10.522727272727273, 6.126315789473685, 13.517241379310345, 10.25]

```

5. Matrices : Operaciones entre matrices

En el caso de las matrices, habría que definirlo en calculadora.thrift usando `list<list<TIPO>>`.

Así pues, definimos una matriz de double (que sería un *array* de *arrays* de **double**) y algunas operaciones para las que usar dos matrices.

```

list<double> restaVectores(1: list<i32> vector1, 2: list<i32>
vector2),
list<double> productoVectores(1: list<i32> vector1, 2: list<i32>
vector2),
list<double> cocienteVectores(1: list<i32> vector1, 2: list<i32>
vector2),

```

En el servidor, como hay que generar una nueva matriz, se debe reservar la memoria. Por ejemplo, al sumar matrices:

```

def sumaMatrices(m1,m2)
total = Array.new(m1.length){Array.new(m2.length)}
for i in 0..m1.length-1 do
  for j in 0..m2.length-1 do
    total[i][j] = m1[i][j] + m2[i][j]
  }
end

```

```

    end
end

return total
end

```

En el cliente, hay que leer la matriz desde archivo. En Python se ejecuta con la orden **python3 cliente.py -m2 <archivo1> <operación> <archivo2>**:

```

ficheroAcumulado1 = argv[2]
#...
matriz1 = []
#...
with open(argv[2]) as f:
    for linea in f:
        linea_anadir = [int(x) for x in linea.split()]
        matriz1.append(linea_anadir)

```

```

ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-rb$ ruby servidor.rb
Iniciando servidor...
ping()
ping()
ping()

ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -m2 ../input_files/matriz1.txt + ../input_files/matriz2.txt
hacemos ping al server
[[10, 82, 39, 79], [54, 19, 91, 73], [26, 51, 20, 62], [64, 100, 49, 18]] + [[21, 19, 91, 99], [41, 57, 83, 76], [15, 31, 33, 16], [70, 59, 82, 84]] = [[31.0, 101.0, 130.0, 178.0], [95.0, 76.0, 174.0, 149.0], [41.0, 82.0, 53.0, 78.0], [134.0, 159.0, 131.0, 102.0]]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -m2 ../input_files/matriz1.txt - ../input_files/matriz2.txt
hacemos ping al server
[[10, 82, 39, 79], [54, 19, 91, 73], [26, 51, 20, 62], [64, 100, 49, 18]] - [[21, 19, 91, 99], [41, 57, 83, 76], [15, 31, 33, 16], [70, 59, 82, 84]] = [[-11.0, 63.0, -52.0, -20.0], [13.0, -38.0, 8.0, -3.0], [11.0, 20.0, -13.0, 46.0], [-6.0, 41.0, -33.0, -66.0]]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$ python3 cliente.py -m2 ../input_files/matriz1.txt x ../input_files/matriz2.txt
hacemos ping al server
[[10, 82, 39, 79], [54, 19, 91, 73], [26, 51, 20, 62], [64, 100, 49, 18]] x [[21, 19, 91, 99], [41, 57, 83, 76], [15, 31, 33, 16], [70, 59, 82, 84]] = [[9687.0, 10734.0, 15481.0, 14482.0], [8388.0, 9237.0, 15480.0, 14378.0], [7277.0, 7679.0, 12343.0, 11978.0], [7439.0, 9497.0, 17217.0, 16232.0]]
ivan@Arcadia-TUF:/mnt/c/Users/Ivan/Desktop/Clase/DSD/Practica/P1b/src/gen-py$

```

7. Conclusiones

La llamada a procedimiento remoto podría permitir realizar tareas complejas que podrían requerir procesadores potentes usando un PC cualquiera como cliente. Además, a diferencia de Sun RPC, permite usar sus funciones y stubs en varios lenguajes, permitiendo así una comodidad e integración con otras tecnologías, como (Node)JS o Ruby (on Rails). Si junto con ello permitimos una distribución de la potencia de cómputo, acabaríamos con una buena versatilidad