Cloud computing

Tema 2. Fundamentos (Parte I)

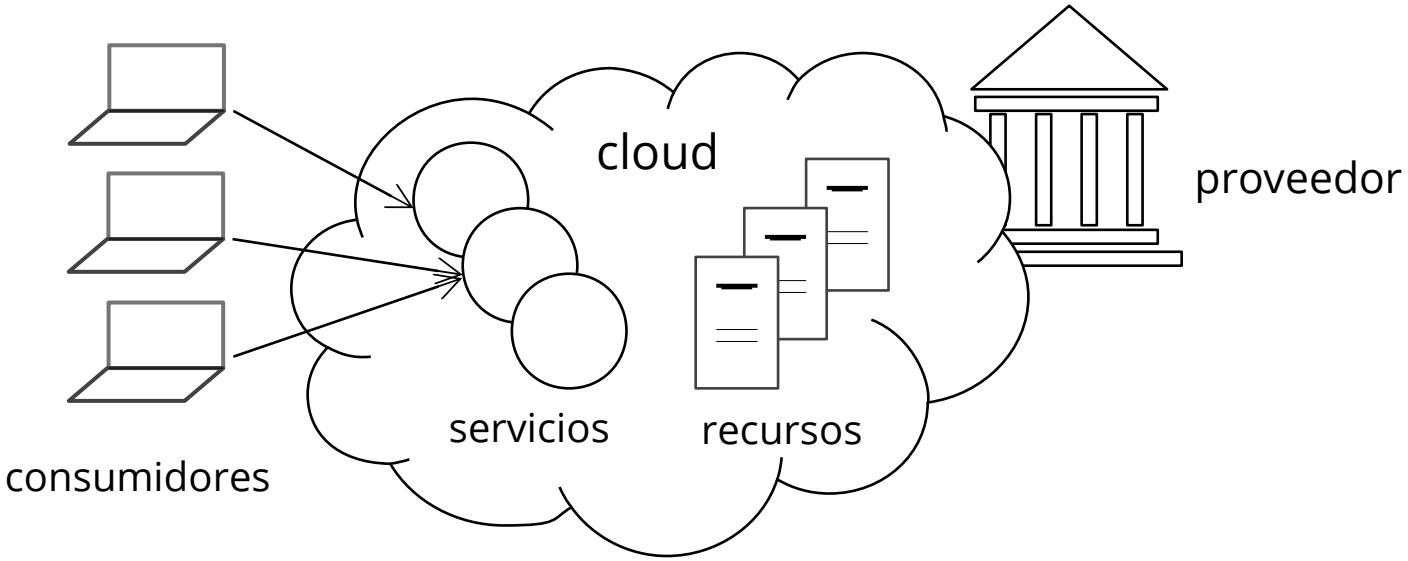
Contenido

- Introducción
- SOA
  - Servicios
  - Principios de diseño
  - Servicios RESTful
- Virtualización
  - Virtualización hardware
  - Libvirt
- Contenedores
  - Docker

Introducción

El modelo

- Cloud computing es un modelo computacional que proporciona recursos computacionales a través de servicios



Introducción

El modelo

- Cloud computing es un modelo computacional que proporciona recursos computacionales a través de servicios
- Los recursos computacionales se implementan con técnicas de virtualización
- Los servicios se organizan en arquitecturas orientadas a servicios (SOA)
- La virtualización y las arquitecturas SOA son 2 tecnologías esenciales (key enablers) para habilitar cloud computing

SOA

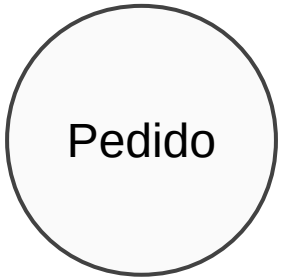
¿Qué es?

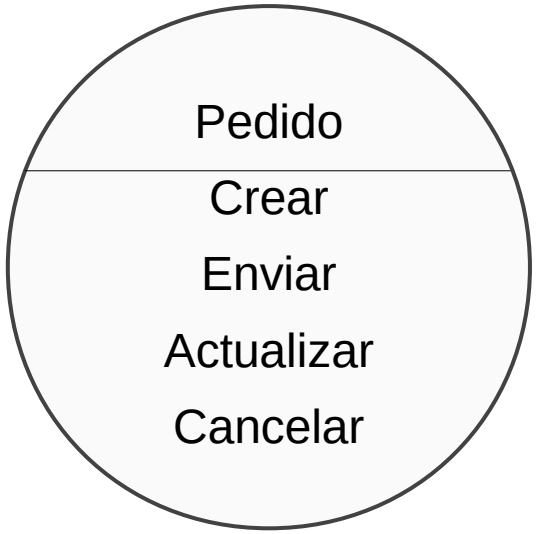
- En cloud computing los recursos computacionales son accedidos a través de servicios
- Arquitecturas SOA: romper sistema en subsistemas independientes, llamados servicios
- Cada servicio publica una interfaz
- Existen distintas aproximaciones y tecnologías: servicios web, REST, gRPC, GraphQL, ...

Orientación a servicios

Servicios

- Fragmento de software que ofrece una colección de capacidades, que se describen en un contrato de servicio
- Las capacidades están relacionadas por un contexto funcional común (cohesión)

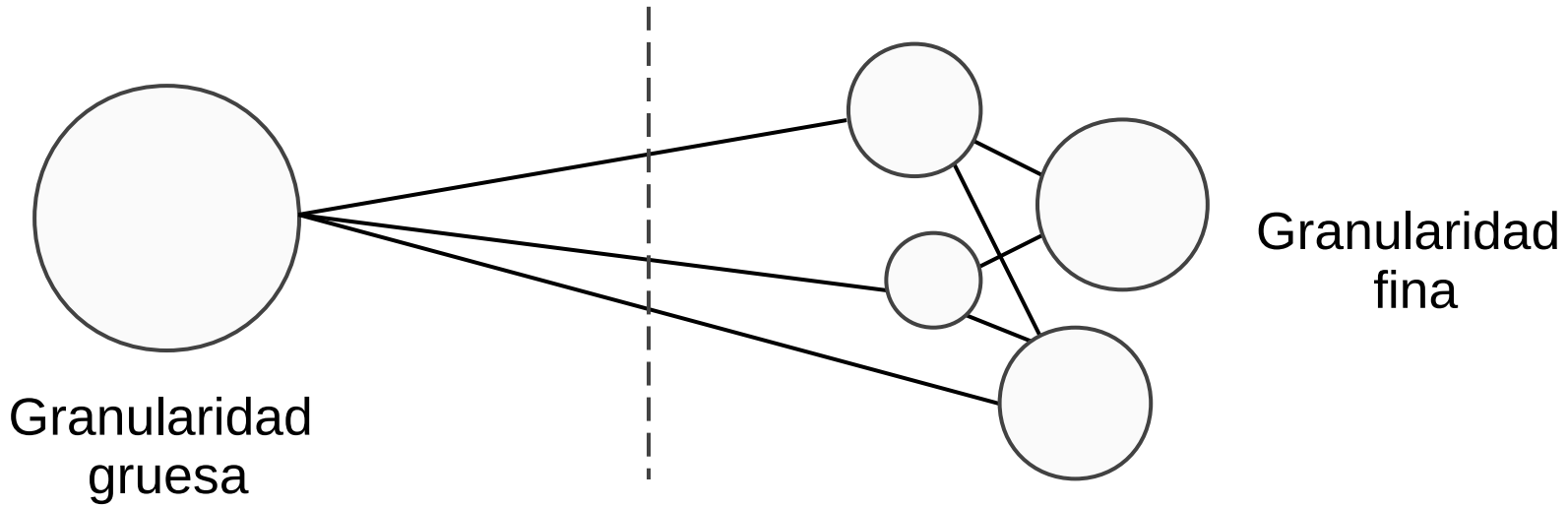




Orientación a servicios

Servicios

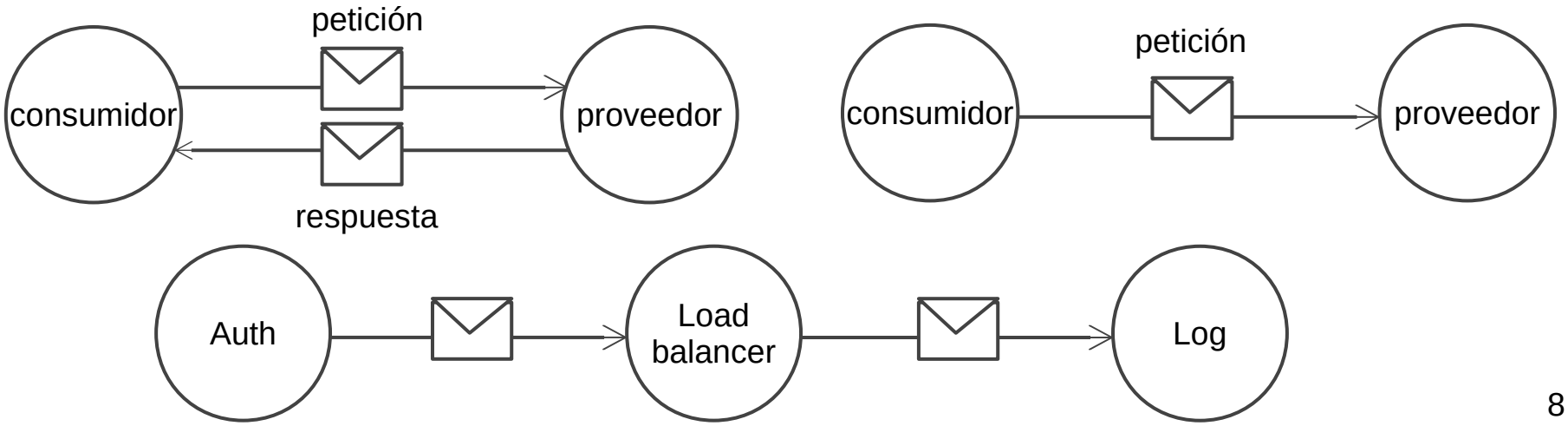
- Granularidad fina vs gruesa: funciones muy concretas y acotadas vs con gran alcance
- Se recomienda granularidad gruesa para simplificar la arquitectura y optimizar comunicaciones



Orientación a servicios

Servicios

- Consumidor vs proveedor
- Comunicación (síncrona vs asíncrona) por paso de mensajes
- Comunicación directa o mediante intermediarios (agentes)



Diseño orientado a servicios

Principios de diseño

- Prácticas generalizadas y aceptadas para obtener arquitecturas SOA con características deseables
- Se identifican 8 principios interrelacionados

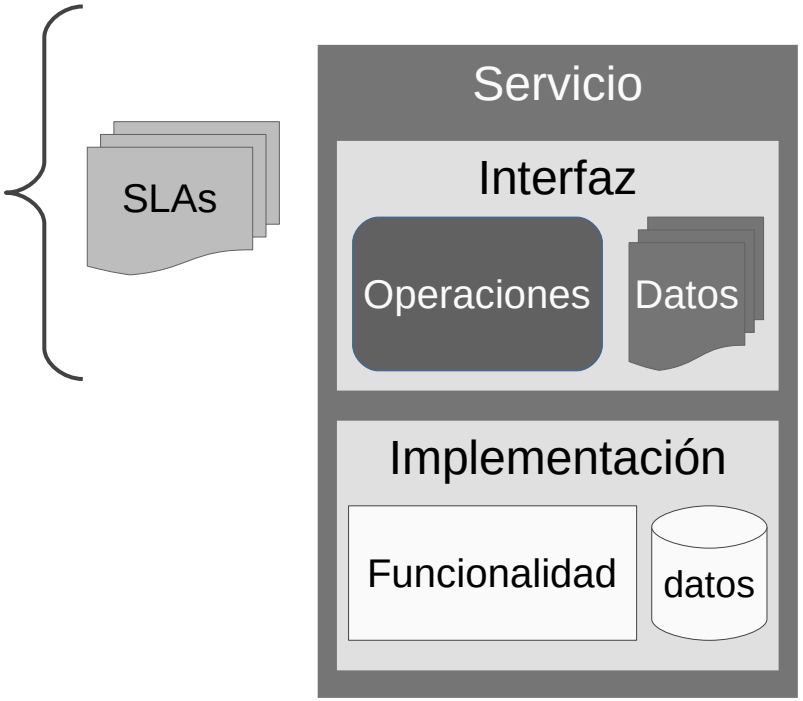
1. Contrato de servicio estándar
2. Bajo acomplamiento
3. Abstracción
4. Reusabilidad
5. Autonomía
6. Sin estado
7. Descubrimiento
8. Composición

Diseño orientado a servicios

Principios de diseño (8)

1. Contrato de servicio estándar

- Debe describir las capacidades, limitaciones, interfaz (API), garantías de servicio (QoS), …



Diseño orientado a servicios

Principios de diseño (8)

2. Bajo acoplamiento

- El contrato desacopla interfaz de implementación
- Además, no impone dependencias sobre el consumidor, ni sobre la implementación
- El servicio evoluciona de manera independiente

3. Abstracción

- El contrato no contiene detalles de implementación
- Se favorece la reutilización y evolución independiente de la implementación

Diseño orientado a servicios

Principios de diseño (8)

4. Reusabilidad

- Las capacidades expuestas son genéricas y pueden reutilizarse en distintos procesos/tecnologías
- Servicios de entidad/utilidad vs de tarea

5. Autonomía

- Auto-gobierno, control sobre el entorno, recursos, sin dependencias externas
- El servicio es más predecible y fiable, evoluciona de manera independiente

Diseño orientado a servicios

Principios de diseño (8)

6. Sin estado

- Delegar almacenamiento a entidad externa (aumenta acoplamiento y reduce autonomía)
- Minimiza el consumo de recursos y favorece escalabilidad

7. Descubrimiento

- Contratos con metadatos para habilitar descubrimiento en repositorios

Diseño orientado a servicios

Principios de diseño (8)

8. Composición

- Fácilmente agregables en servicios compuestos
- Depende en gran medidad del resto de principios

Servicios RESTful

¿Qué es?

- Los servicios RESTful surgen como alternativa a los servicios web (SOAP), basados en XML
- SOAP es muy potente, pero muy complejo, y poco eficiente
- Servicios RESTful:
  - Permiten gestionar colección recursos remotos
  - Simple, eficiente y con tecnologías estándar
  - Verifican los objetivos de las arquitecturas REST

Servicios RESTful

Hoja de ruta

- Principios de arquitectura
- El protocolo HTTP
- El contrato de servicio
- Implementación con Python
- Consumo de servicios

Servicios RESTful

Principios de arquitectura

- REST: Representational State Transfer
- Aparece por primera vez en la tesis doctoral de Roy Fielding
- Originalmente se refería a un conjunto de principios de arquitectura, aprendidos de WWW
- Actualmente se utiliza para describir cualquier interfaz remota que use HTTP como protocolo de comunicación

Servicios RESTful

Principios de arquitectura

- Cliente-servidor
- Sin estado
- Cache
- Interfaz uniforme
- Sistema a capas
- Código bajo demanda

Servicios RESTful

Principios de arquitectura

- Un servicio RESTful verifica los principio REST, y sigue las siguientes reglas:
  - Envuelve una colección de recursos. Cada recurso posee un identificador-URI único
  - Recursos se manipulan a través de representaciones: los datos son documentos
  - Múltiples representaciones por recurso (XML,JSON,...)
  - Recursos accedidos por operaciones CRUD
  - Proveedor sin estado

Servicios RESTful

Principios de arquitectura

- Propiedades de las arquitecturas REST:
  - Rendimiento
  - Escalabilidad
  - Simplicidad
  - Modificabilidad
  - Visibilidad
  - Portabilidad
  - Fiabilidad

Servicios RESTful

HTTP

- Protocolo de transporte habitual en RESTful
- Protocolo de comunicación sin estado que permite la transferencia/manipulación de recursos en Internet
- Los recursos se representan por medio de URLs (Uniform Resource Locator)

esquema://máquina:puerto/path?query#fragmento

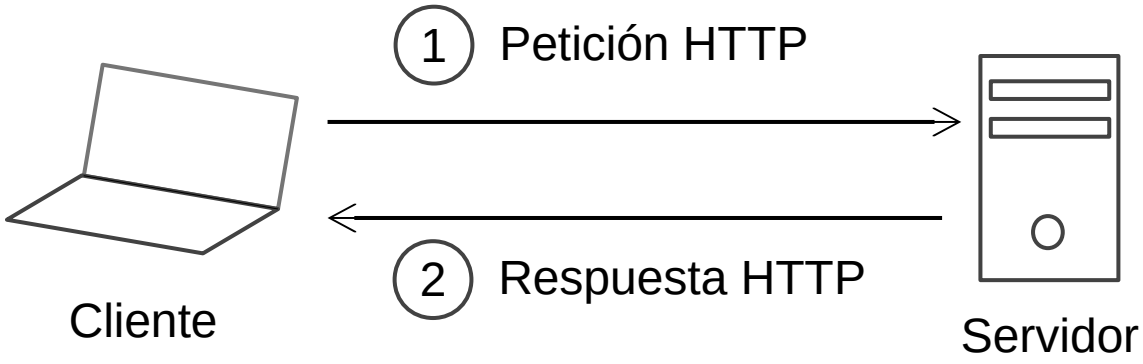
http://ejemplo.com/data/api/user/username?kind=alumno

- Soporta distintos tipos de operaciones sobre recursos: GET, POST, PUT, DELETE, etc.

Servicios RESTful

HTTP

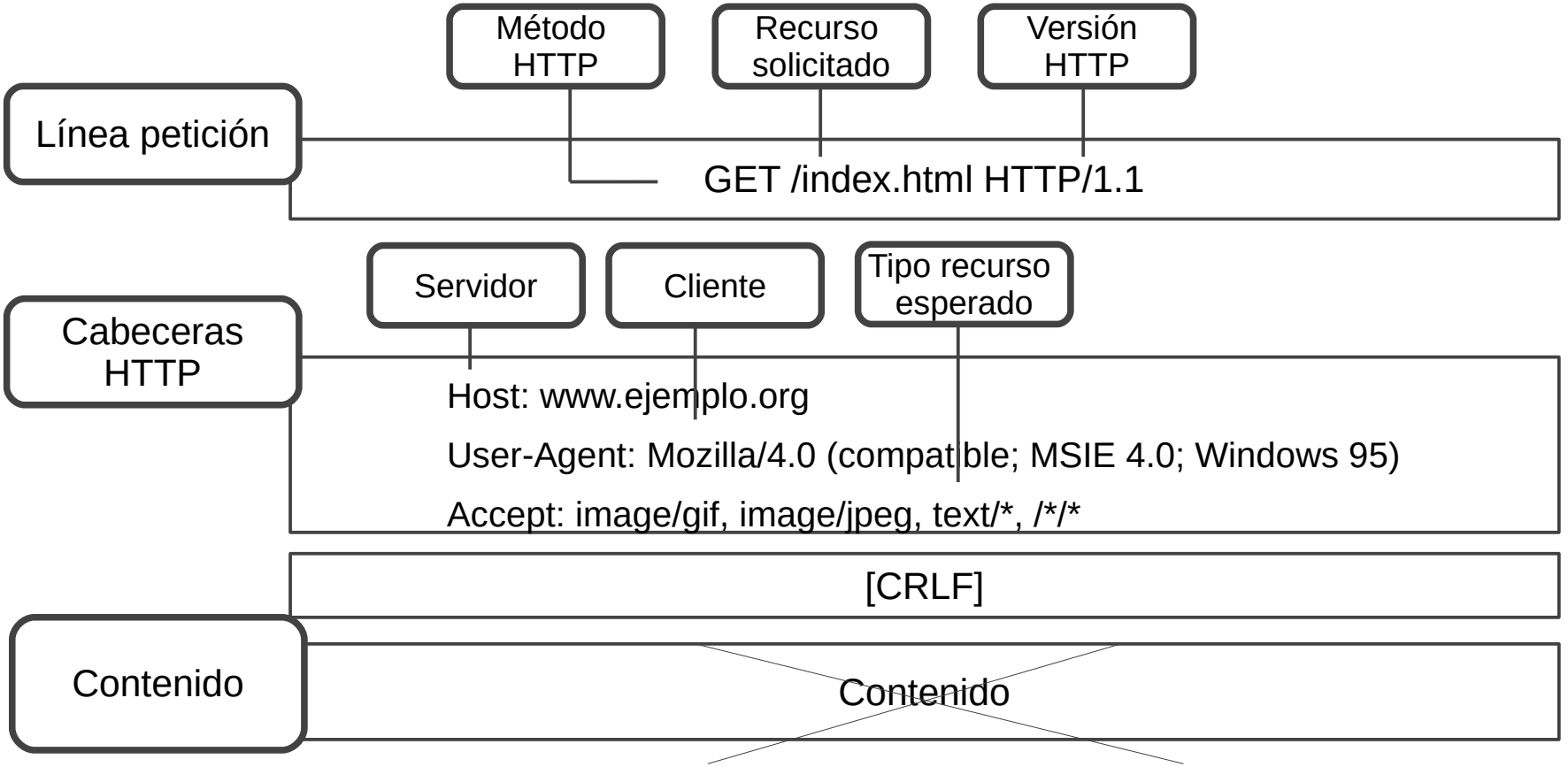
- Petición/respuesta HTTP



Servicios RESTful

HTTP

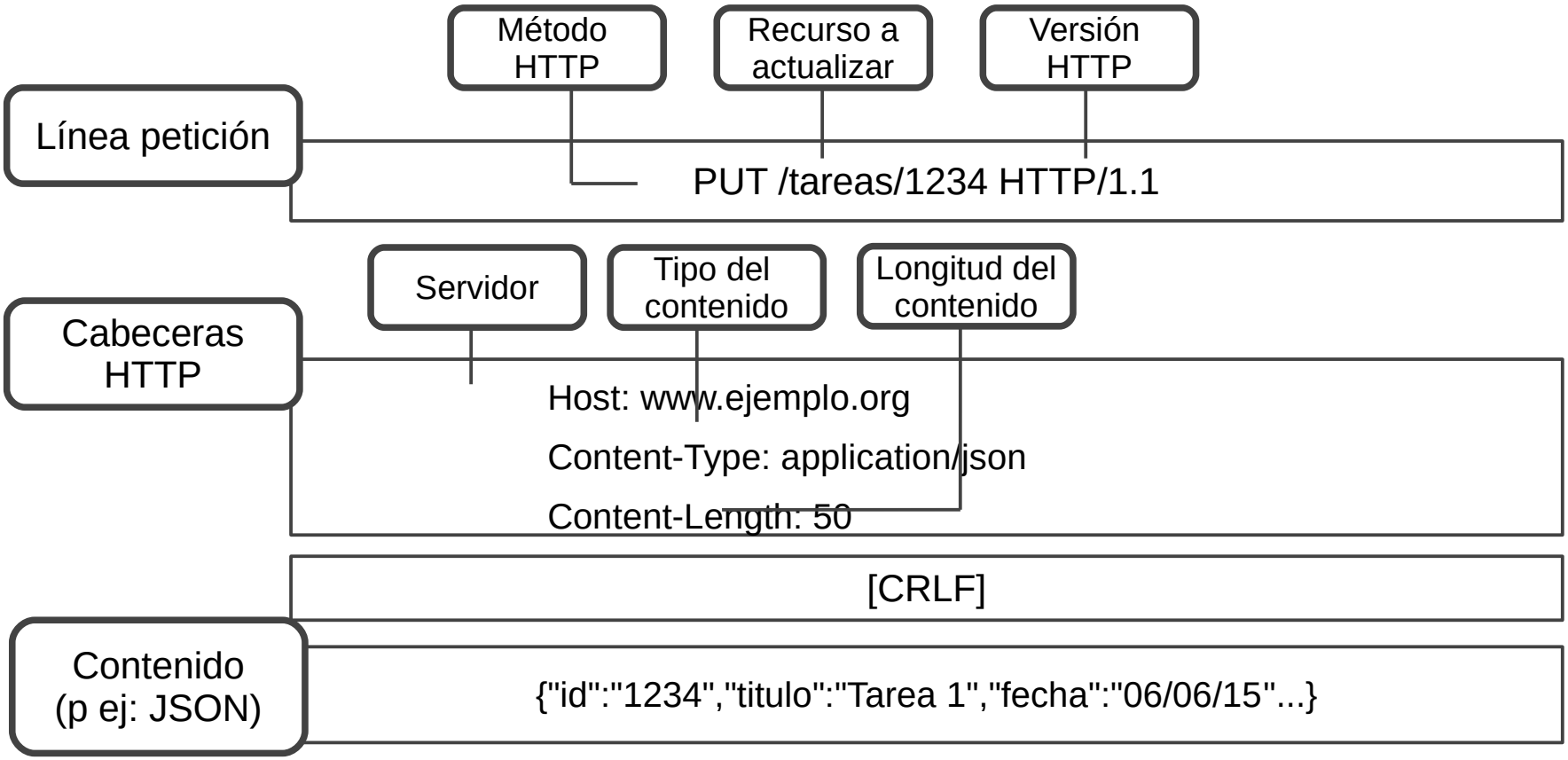
- Petición HTTP de recuperación



Servicios RESTful

HTTP

- Petición HTTP de actualización



Servicios RESTful

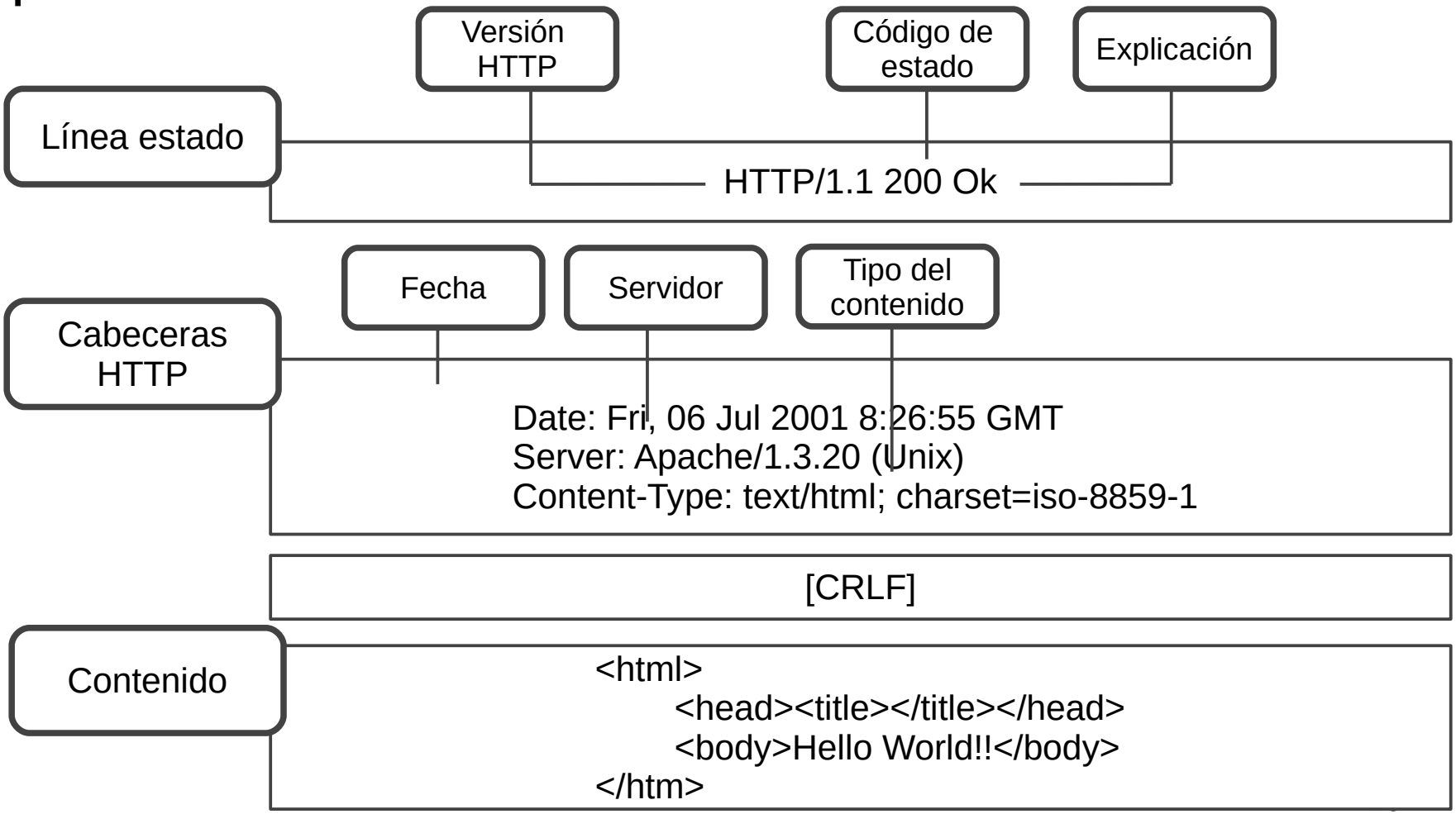
HTTP

- Cabeceras soportadas en una petición HTTP
  - Accept, Accept-Charset, Accept-Encoding, Accept-Language, Accept-Datetime, Authorization, Connection, Cookie, Content-Length, Content-MD5, Content-Type, Date, Expect, From, Host, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Max-Forwards, Origin, Proxy-Authorization, User-Agent, ...

Servicios RESTful

HTTP

- Respuesta HTTP



Servicios RESTful

HTTP

- Códigos de error HTTP:
  - 1xx (información)
  - 2xx (éxito)
  - 3xx (redirección)
  - 4xx (error causado por cliente)
  - 5xx (error causado por servidor)

Servicios RESTful

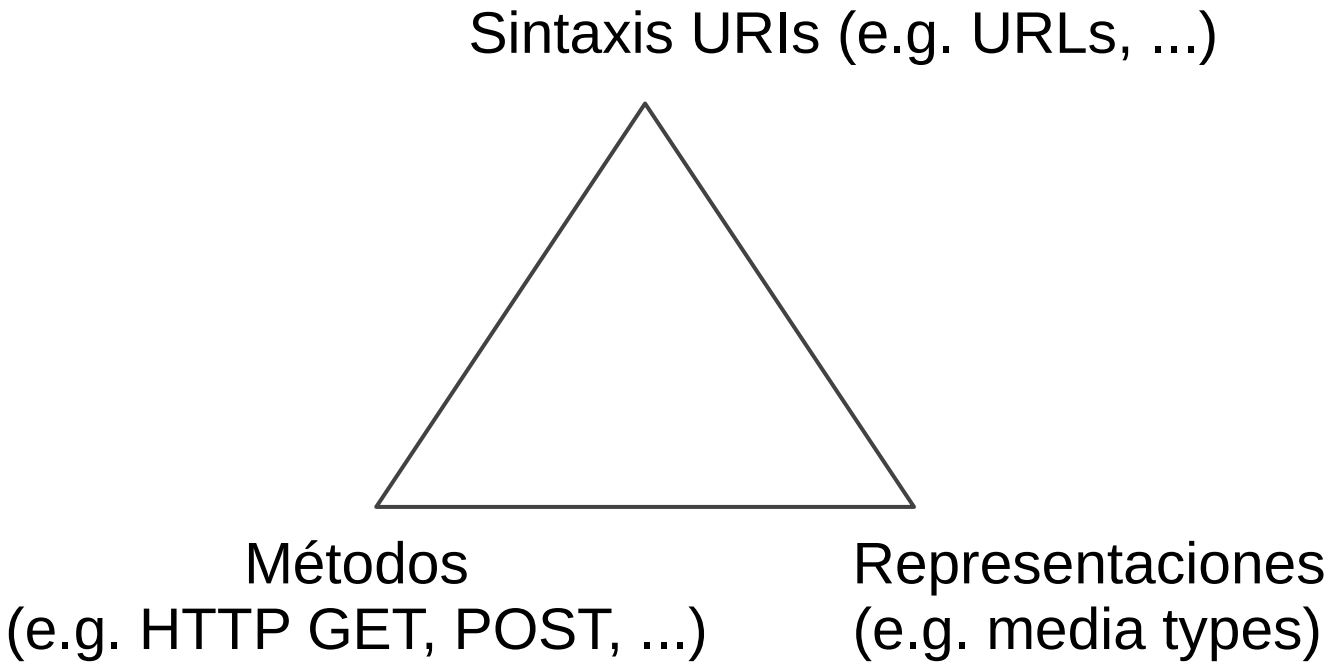
HTTP

- Cabeceras soportadas en una respuesta HTTP
  - Access-Control-Allow-Origin, Accept-Patch, Accept-Ranges, Age, Allow, Cache-Control, Connection, Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-MD5, Content-Range, Content-Type, Date, Expires, Last-Modified, Link, Location, Proxy-Authenticate, Retry-After, Server, Set-Cookie, Status, WWW-Authenticate, ...

Servicios RESTful

Contrato de servicio

- Queda determinado por 3 elementos



Servicios RESTful

Contrato de servicio > Identificadores

`{scheme}://{authority}{path}?{query}`

- Los IDs se construyen a partir de una URI base
- Nombres, no acciones
- Minúsculas, ‘-’ en lugar de ‘_’
- Subcolecciones con ‘/’
- Filtrado con ‘?’
- Versiones

```
http://www.example.org/users/{userId}
http://www.example.org/users/{userId}/accounts/{accountId}
http://www.example.org/users?name=Pepe
http://www.example.org/v1/users
```

Servicios RESTful

Contrato de servicio > Métodos

- Determinan las operaciones sobre los recursos
- Genéricas CRUD (Create-Read-Update-Delete)
- Se reaprovechan los métodos HTTP
  - GET: recupera un recurso o colección
  - POST: crea un nuevo recurso en colección
  - PUT: actualiza un recurso en colección
  - DELETE: elimina un recurso de colección

Servicios RESTful

Contrato de servicio > Representaciones

- Determina los tipos de datos (*media types*) que utilizan las operaciones del contrato

```
type/subtype [; parameter]
```

- En HTTP se usan las cabeceras
  - Accept: en peticiones, determina el tipo esperado
  - Content-Type: en peticiones/respuestas, determina el tipo de contenido

```
Accept: text/html; charset=UTF-8
Content-Type: application/json
```

Servicios RESTful

Contrato de servicio > Representaciones

- JSON (JavaScript Object Notation)
  - Formato de datos basado en texto, eficiente, para almacenamiento/trasferencia de datos
  - Soporta los tipos de datos: number, boolean, string, null, object, array

```
conv = {'id': 1, 'title': 'family', 'owner': True, 'date': None,
'mesages': [{'id': 1, 'content': 'hello'}, {'id': 2, 'content':
'bye'}]}
```

```
import json
the_json=json.dumps(conv)
```

```
conv = json.loads(the_json)
```

```
'{"id": 1, "title": "family", "owner": true, "date": null,
"mesages": [{"id": 1, "content": "hello"}, {"id": 2, "content":
"bye"}]}'
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas

`/mistareas/tareas`

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas `/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP

```
Req: HTTP POST /mistareas/tareas
        {"titulo":"Tarea 1","fecha":"25/03/2015"}
Resp: 200/201 – Ok/Created
        {"id":"1234","titulo":"Tarea 1","fecha":"25/03/2015"}
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas `/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP
- HTTP DELETE: elimina una tarea con la url especificada en la petición HTTP

```
Req: HTTP DELETE /mistareas/tareas/1234
Resp: 204 – No Content
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas `/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP
- HTTP DELETE: elimina una tarea con la url especificada en la petición HTTP
- HTTP PUT: modifica una tarea. La tarea y los datos se especifican en la petición HTTP

```
Req: HTTP PUT /mistareas/tareas/1234
        {"titulo":"Tarea 1.1","fecha":"25/03/2015"}
Resp: 200 – Ok
        {"id":"1234","titulo":"Tarea 1.1","fecha":"25/03/2015"}
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas `/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP
- HTTP DELETE: elimina una tarea con la url especificada en la petición HTTP
- HTTP PUT: modifica una tarea. La tarea y los datos se especifican en la petición HTTP
- HTTP GET: recupera la/s tarea/s que corresponde/n con la url especificada en la petición HTTP

```
Req: HTTP GET /mistareas/tareas
```

```
Resp: 200 – Ok
  [{"id":"1","titulo":"Tarea 1"},
   {"id":"2","titulo":"Tarea 2"}]
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas `/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP
- HTTP DELETE: elimina una tarea con la url especificada en la petición HTTP
- HTTP PUT: modifica una tarea. La tarea y los datos se especifican en la petición HTTP
- HTTP GET: recupera la/s tarea/s que corresponde/n con la url especificada en la petición HTTP

```
Req: HTTP GET /mistareas/tareas/123
```

```
Resp: 200 – Ok
  {"id":"1","titulo":"Tarea 1"}
```

Servicios RESTful

Contrato de servicio > Ejemplo

Servicio que sirva tareas

`/mistareas/tareas`

- HTTP POST: se crea una nueva tarea con el contenido especificado en la petición HTTP
- HTTP DELETE: elimina una tarea con la url especificada en la petición HTTP
- HTTP PUT: modifica una tarea. La tarea y los datos se especifican en la petición HTTP
- HTTP GET: recupera la/s tarea/s que corresponde/n con la url especificada en la petición HTTP

```
Req: HTTP GET /mistareas/tareas?
  titulo=xxx&fecha=xxx
```

```
Resp: 200 – Ok
  [{"id":"1","titulo":"Tarea 1"},
   {"id":"2","titulo":"Tarea 2"}]
```

Implementación servicios RESTful

Flask

- https://flask.palletsprojects.com/
- Framework minimalista para construir aplicaciones web en el servidor con Python
- Posee múltiples extensiones
- Instalación en entorno virtual

```
> python3 -m venv venv
> source venv/bin/activate
(venv) > pip install flask
```

Implementación servicios RESTful

Flask

- Ejemplo rápido

hello.py

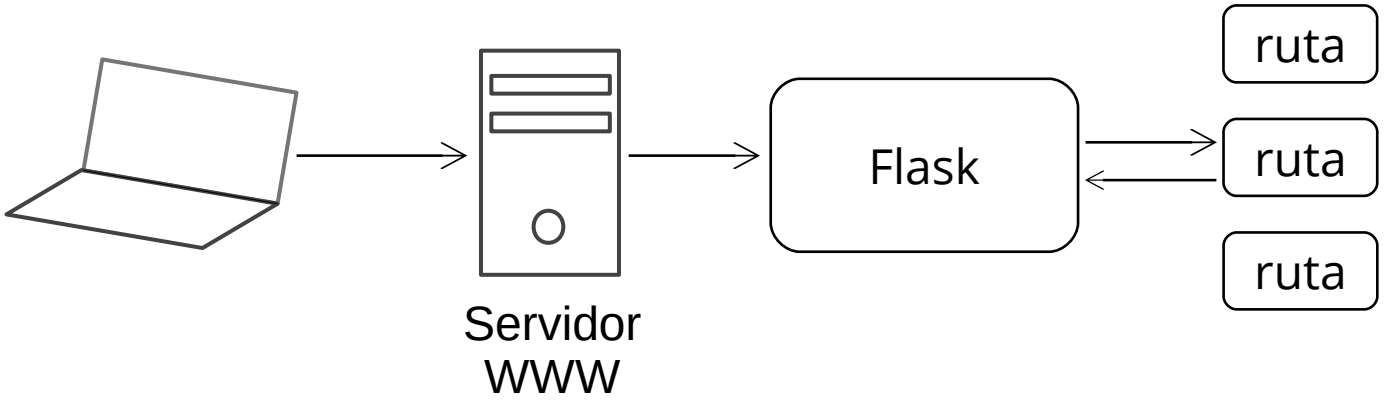
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
  return '<h1>Hello World!</h1>'
```

```
(venv) > export FLASK_APP=hello.py
(venv) > export FLASK_DEBUG=1
(venv) > flask run

(venv) > flask --app hello --debug run
```

Implementación servicios RESTful

Rutas

- Flask mapea URLs a funciones que generan respuesta: rutas



Implementación servicios RESTful

Rutas

- Se definen utilzando el decorador @app.route('url' [,opts])
- Todas las rutas registradas están disponibles en app.url_map

```
@app.route('/hello')
def hello(): return '<h1>Hello World!</h1>'
@app.route('/bye')
def bye(): return '<h1>Bye World!</h1>'
```

Implementación servicios RESTful

Rutas > métodos HTTP

- Por defecto una ruta sirve peticiones HTTP GET/HEAD/ OPTIONS
- Es posible servir otras peticiones HTTP con methods

```
@app.route('/', methods=['GET','POST','PUT','DELETE'])
def index():return '<h1>Hello World!</h1>'
```

Implementación servicios RESTful

Rutas > rutas dinámicas (paramétricas)

- Se define un parámetro <param> en la URL
- Representa una colección de URLs
- El parámetro se recibe como string en la función manejadora
- El parámetro se puede convertir a otro tipo

```
@app.route('/user/<name>')
def user(name):
  return '<h1>Hello, {}!</h1>'.format(name)
@app.route('/user/<int:userId>')
def user(userId):
  return f'User {userId}'
```

Implementación servicios RESTful

Petición

- Flask crea un contexto de ejecución para cada petición
- flask.request contiene toda la info de la petición actual

```
from flask import request
@app.route('/')
def index():
  request.method
  request.url
  request.headers.get('User-Agent')
  request.args.get('param')
  request.get_data()
  request.get_json()
```

Implementación servicios RESTful

Respuesta

- Flask crea una respuesta a partir del valor devuelto por la función manejadora
- str → text/html; dict, list → application/json

```
@app.route('/')
def index():return '<h1>Hello World!</h1>'

@app.route('/user/<name>', methods=['POST'])
def user(name): return {'name': name}
```

Implementación servicios RESTful

Respuesta

- Es posible controlar cualquier aspecto de la respuesta, devolviendo (resp,status), (resp,status,headers)
- Más control utilizando flask.make_response()

```
@app.route('/')
def index(): return '<h1>Bad Request</h1>', 400

from flask import make_response
@app.route('/')
def index():
  response = make_response('<h1>cookie!</h1>')
  response.set_cookie('answer', '42')
  return response
```

Implementación servicios RESTful

Errores

- Flask detecta excepciones y devuelve errores automáticamente
- Más control utilizando @app.errorhandler(status)
- Para abortar petición en curso abort(status)

```
@app.errorhandler(404)
def page_not_found(e):
    return '<h1>Not found!</h1>', 404

from flask import abort
@app.route("/")
def index():
  if error: abort(404)
```

Implementación servicios RESTful

Implementación API RESTful

- Conjunto de rutas que implementan API RESTful

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

@app.route('/myapp/tasks')
def listTasks(): ...
@app.route('/myapp/tasks/<taskId>')
def getTaskById(taskId): ...
@app.route('/myapp/tasks', methods=['POST'])
def addTask(): ...
@app.route('/myapp/tasks/<taskId>', methods=['PUT'])
def updateTask(taskId): ...
@app.route('/myapp/tasks/<taskId>',
methods=['DELETE'])
def deleteTask(taskId): ...
```

Implementación servicios RESTful

Implementación API RESTful

- Cada manejador de ruta
  1. Procesa la información de entrada
  2. Genera la información de salida

Implementación servicios RESTful

Procesar información de entrada

- Si es HTTP GET:
  - La URL en request.url ó request.path
  - Datos de la query en request.query_string ó request.args

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/myapp/users', methods=['GET'])
def users():
  name = request.args['name']
  surname = request.args['surname']
```

Implementación servicios RESTful

Procesar información de entrada

- Si NO es HTTP GET:
  - Datos en cuerpo del mensaje con request.get_data() ó request.get_json()

```
POST /myapp/tasks HTTP/1.1
Host: www.ejemplo.org
Content-Type: application/json
Content-Length: 50

{"id": "1234","title":"Tarea 1","date":"06/06/15"...}
```

Implementación servicios RESTful

Procesar información de entrada

- Si NO es HTTP GET:
  - Datos en cuerpo del mensaje con request.get_data() ó request.get_json()

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/myapp/tasks', methods=['POST'])
def addTask():
  task = request.get_json()
  print(f"id:{task['id']}")
  print(f"titulo:{task['title']}")
  return 'OK', 200
```

Implementación servicios RESTful

Procesar información de entrada

- Las rutas dinámicas son muy comunes para consultas

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/myapp/tasks/<taskId>')
def getTaskById(taskId):
  print(f'id:{taskId}')
```

Implementación servicios RESTful

Generar información de salida

- Si es una consulta se devuelve el objeto

```
@app.route('/myapp/tasks/<taskId>')
def getTaskById(taskId):
  return {'id': taskId}
```

- Si es una eliminación, no se devuelve contenido

```
@app.route('/myapp/tasks/<taskId>',methods=['DELETE'])
def removeTask(taskId):
  removeTaskInDB(taskId)
  return '', 204
```

Consumo de servicios RESTful

Consumo de servicios

- Únicamente es necesario contar con un cliente HTTP
- Peticiones HTTP GET: navegador web
- Otras peticiones: Postman, Insomnia, …
- En Python:
  - Módulo http.client
  - HTTP para humanos con requests

Consumo de servicios RESTful

Consumo de servicios > requests

- Instalar

```
> pip install requests
```

- Petición

```
import requests
resp = requests.get(url,params=None)
resp = requests.post(url,data=None,json=None)
resp = request.put(url,data=None,json=None)
resp = request.delete(url)
```

Consumo de servicios RESTful

Consumo de servicios > requests

- Respuesta

```
resp.status_code
resp.headers
resp.content
resp.text
resp.json()
```

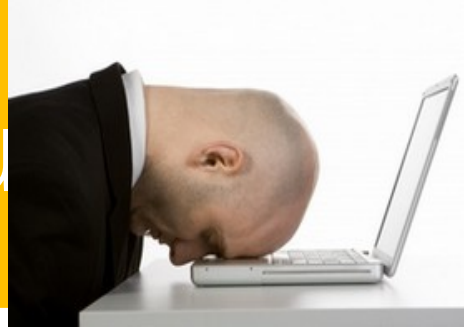
Consumo de servicios RESTful

Consumo de servicios > requests

- Ejemplos

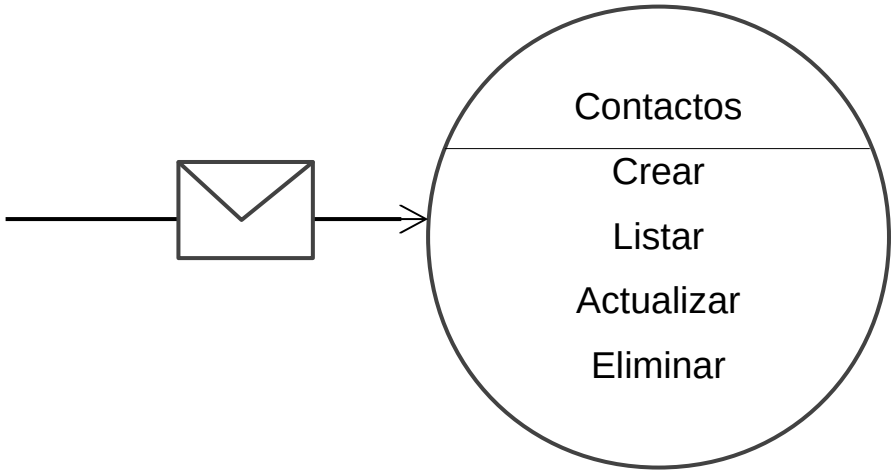
```
import requests
response = requests.get("http://localhost:5000/")
print(response.text)

response = requests.post("http://localhost:5000/users",
             json={"name":"Pepe"})
print(response.json())

response =
   requests.delete("http://localhost:5000/users/pepe")
print(response.status_code)
```

Consumo de servicios RESTfu



Ejercicio 1

- Servicio RESTful que gestione contactos
- Cliente CLI que se comunique con el servicio




TO BE
CONTINUED...