Hyper **T**ext **T**ransfer **P**rotocol (HTTP)

Protocolos de comunicación en redes

“sistema de reglas que permiten que dos o más entidades de un sistema se comuniquen”

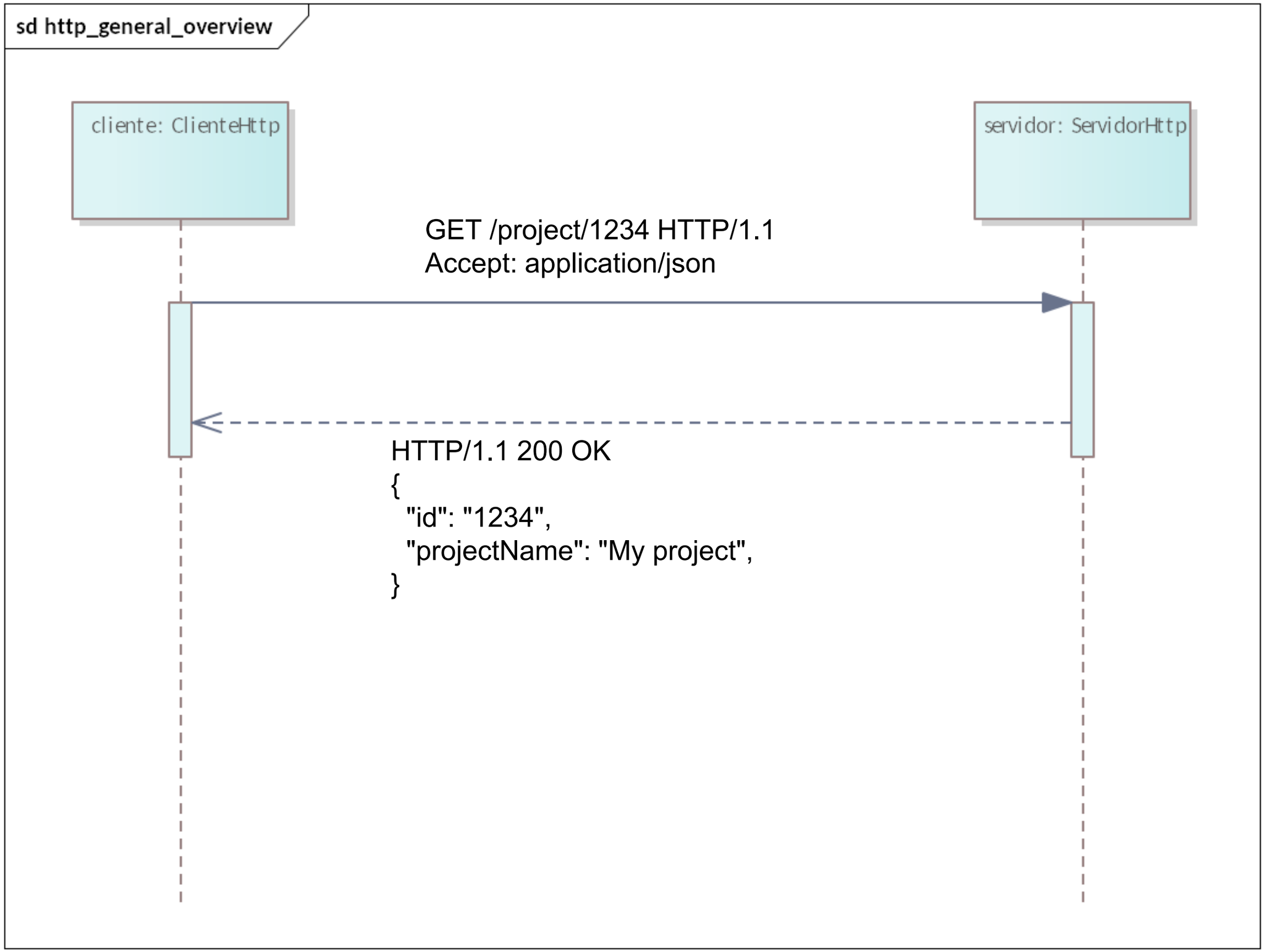
Protocolo de comunicación

Introducción

- HTTP es un protocolo de la **capa de aplicación** para transmitir documentos de hipermedia.
- Base para **solucionar problemas de integración** y diseño de los sistemas en red y sus componentes.

- HTTP es una de las tecnologías centrales de la web.
- Los navegadores realizan múltiples solicitudes HTTP para cargar una página web.
- El protocolo HTTP comenzó como un protocolo simple basado en texto.
- HTTP se ha vuelto más complejo, pero el formato básico basado en texto no ha cambiado en los últimos 20 años.
- HTTPS cifra los mensajes HTTP estándar.
- Hay varias herramientas disponibles para ver y enviar mensajes HTTP.

- HTTP es un protocolo cliente-servidor:
  1. Un cliente envía una solicitud a un servidor.
  2. El servidor determina si puede servir la solicitud con la información dada.
  3. El servidor retornará una respuesta que incluye:
     - Un código indicando éxito o fracaso
     - Una carga útil (*payload*) que contiene la información solicitada o detalles sobre el error


sd http_general_overview
cliente: ClienteHttp
servidor: ServidorHttp
GET /project/1234 HTTP/1.1
Accept: application/json
HTTP/1.1 200 OK
{
"id": "1234",
"projectName": "My project",
}

Elementos que componen a HTTP

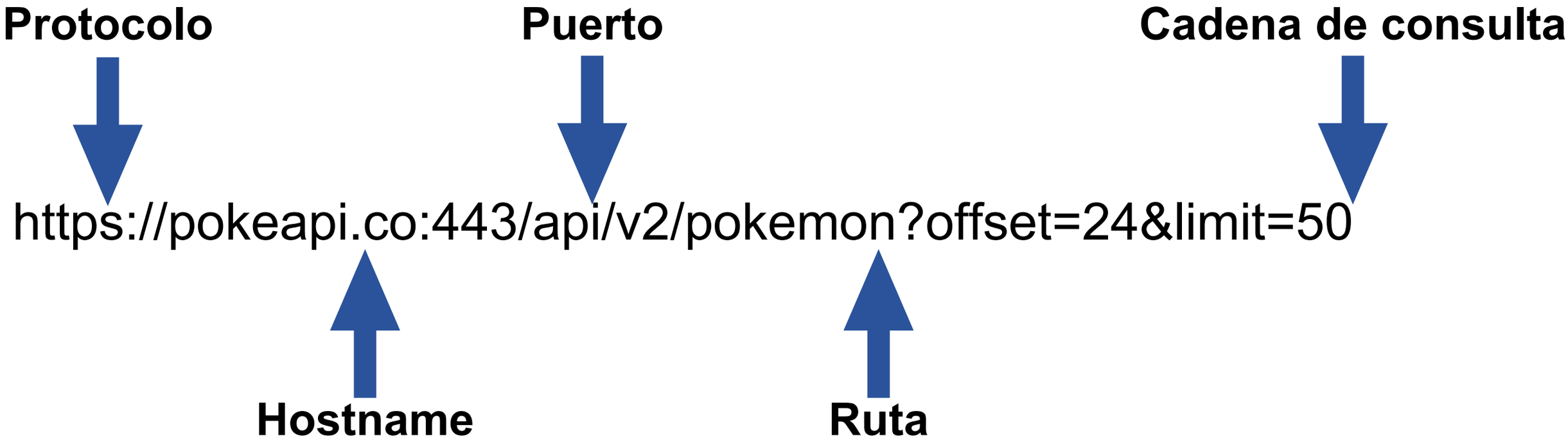
- El ***U**niform **R**esource **L**ocator* (URL) donde se envía la solicitud.
- El método HTTP que informa al servidor cómo el cliente desea interactuar con el recurso.
- Los encabezados y cuerpo de la solicitud y respuesta.
- Un código de respuesta que indica si la solicitud se procesó correctamente o si se encontró un error.

Uniform Resource Locator (URL)

- HTTP utiliza un URL como una dirección única donde se encuentran los datos o servicios.
- Las solicitudes se envían a la URL, donde el servidor procesa la solicitud y envía una respuesta al cliente.
  - https://www.uv.mx
  - https://www.google.com

- Un URL se compone de:
  - **Protocolo**: El protocolo que se utilizará, ej. **http** o https
  - **Hostname**: El servidor a contactar, ej. https://api.nasa.gov/
  - **Puerto**: Un número entre **0** a **65,535** que define el proceso en el servidor a donde se enviará la solicitud, ej. **80** para **http** o **443** para **https**

- **Ruta**: La ruta al recurso que se solicita, por ej. /pokemon/charizard. La ruta predeterminada es /
- **Cadena de consulta**: Contiene datos para el servidor. Comienza con un signo de interrogación y pares de nombre=valor con *ampersand* entre ellos.


Protocolo
Puerto
Cadena de consulta
https://pokeapi.co:443/api/v2/pokemon?offset=24&limit=50
Hostname
Ruta

Solicitud HTTP

- Una solicitud está compuesta por: método HTTP, ruta, encabezado y cuerpo del mensaje.
  - **Método HTTP**: Informa al servidor el tipo de interacción que el cliente desea solicitar.
  - **Ruta**: Es la porción de la URL que hace referencia al recurso en el servidor.

- **Encabezado**: Da detalles del cliente y sobre la solicitud.
  - Se compone de campos en formato de nombre:valor.
  - Los encabezados comunes son:
    - **Accept:** Informa al servidor que tipo de contenido soporta el cliente.
      - image/gif, image/jpeg, */*

- **Content-type**: Informa al servidor del tipo de contenido del cuerpo del mensaje de la solicitud.
- **User-Agent**: Cadena de formato libre que indica el tipo de cliente HTTP que realiza la solicitud.
  - Normalmente indica un tipo y versión de navegador, una biblioteca o herramienta de línea de comandos.

    - **Accept-Encoding**: Informa al servidor qué soporte de compresión procesa el cliente. Permite al servidor reducir el tamaño de la respuesta. Ej. gzip

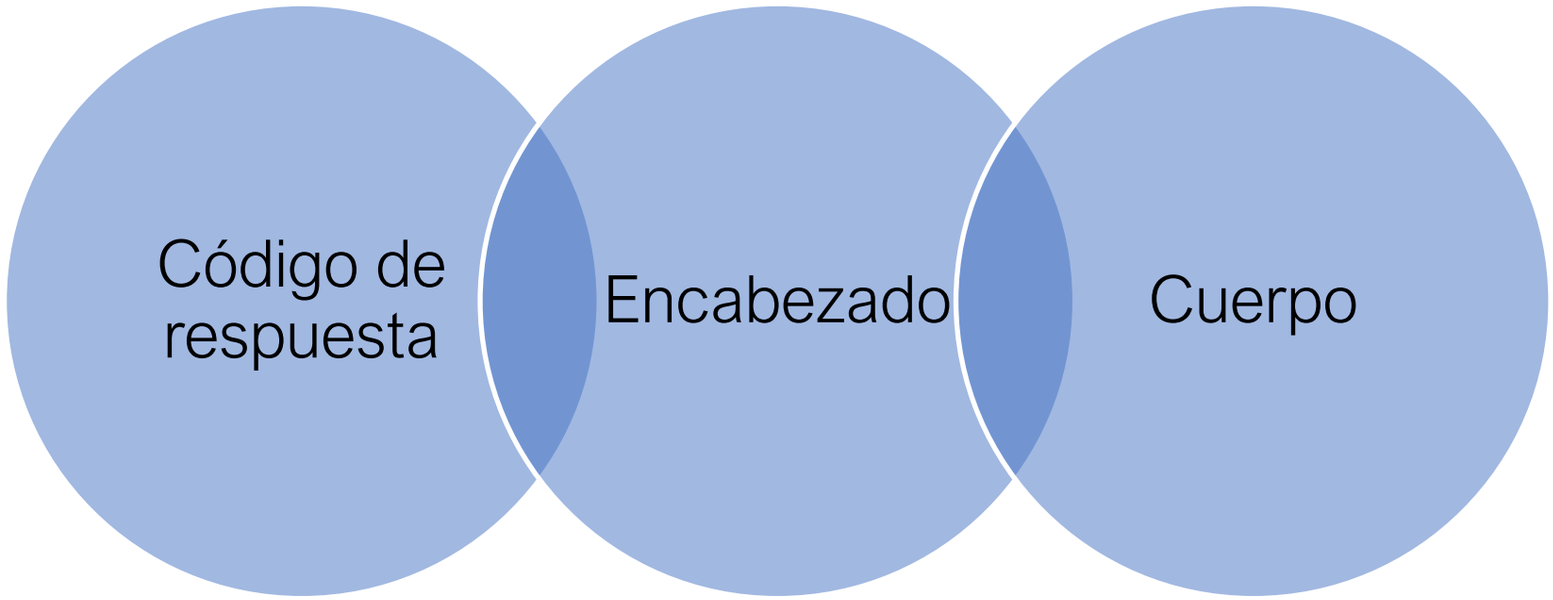
- **Cuerpo del mensaje**: Provee detalles al servidor cuando se envían datos.
    - Pueden ser legibles por el humano o binarios.
    - Para algunas solicitudes puede estar vacío.

```
▶ http -v PUT httpbin.org/put hello=world
PUT /put HTTP/1.1
Accept: application/json, */*;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 18
Content-Type: application/json
Host: httpbin.org
User-Agent: HTTPie/2.6.0

{
    "hello": "world"
}
```

Respuesta HTTP

- Una vez que se recibe una solicitud el servidor la proceso y envía la respuesta.
- Está compuesta por:



- El **código de respuesta** es un número que corresponde a un código de éxito o error que indica si la solicitud pudo cumplirse.
  - Debe estar descrito en la especificación HTTP
  - Solo se permite un código de respuesta por respuesta

- El **encabezado** le dice al cliente detalles sobre el resultado de la solicitud.
  - Se compone de campos de encabezado en formato nombre:valor
  - Los encabezados comunes:
    - **Date**: Fecha de la respuesta.
    - **Content-Location**: La URL completa de la respuesta. Útil si la solicitud resulta en redirecciones que pudieran requerir que el cliente actualice su URL para el recurso.

- **Content-Length**: Longitud en bytes del cuerpo del mensaje de respuesta.
- **Content-Type**: Informa al cliente el tipo de contenido del cuerpo del mensaje.
- **Server**: Una cadena que provee detalles acerca del servidor. El servidor podría proveer pocos o ningún detalle.

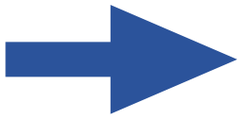
- El cuerpo del mensaje de respuesta proporciona el contenido al cliente.
- Puede ser una página HTML, una imagen o datos en XML, JSON o CSV, como se indica en el encabezado de respuesta Content-Type.

```
▶ http -v www.google.com
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: www.google.com
User-Agent: HTTPie/2.6.0



HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Encoding: gzip
Content-Length: 6612
Content-Type: text/html; charset=ISO-8859-1
Date: Mon, 15 Nov 2021 23:38:19 GMT
Expires: -1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
Set-Cookie: 1P_JAR=2021-11-15-23; expires=Wed, 15-Dec-2021 23:38:19 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=511=t1v5xdXWYI7i_O0qDZY6SyFsx9a73MOISBOW73F-ztU9I4Z2N9tyiSSwjhYomwvlBYbtalvwq7v90zznB7sS5YVFpsie0yvZwMOhePPY33YfG4QTP_Jvf7VEgz32-07lJJ
; domain=.google.com; HttpOnly
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 0
```

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="es-419"><head><meta content="text/html; charset=UTF-8" http-equiv="Conten
mprop="image"><title>Google</title><script nonce="DLHiqBiHZ8rlLKQVfo/1CQ==">(function(){window.google={kEI:'a--SYdD3FdXr-Qbzn7KQBA',kEXPI:'0,202371,11
51224,16114,17444,11240,17572,4859,997,364,9291,3025,2818,14773,4012,978,13228,3847,4192,6430,21822,919,6674,1279,2212,530,149,1103,840,1983,4314,3514
,230,6459,149,13975,4,1252,276,2304,6462,577,4683,2015,1300,12311,4764,2658,6701,656,30,13628,2305,2132,16786,652,1870,3305,2530,4094,3140,4,908,3,354
719,980,2381,2719,3986,2,6033,4148,4073,2,4,4,1697,6074,2124,2444,2577,3124,116,170,268,1110,5525,91,3775,2,4436,3147,545,2172,1929,689,437,5385,2,4,4
121,751,1526,268,225,258,903,454,496,201,341,222,445,60,520,112,399,20,208,10,31,368,111,2,2,1,332,615,93,156,2,1061,1299,1571,112,449,83,1369,434,104
3,2,106,1398,415,778,874,6,3,1,2,4,3,2,7,5,14,6,3,907,3,82,46,411,636,1129,6,2,42,67,217,421,859,2,5530117,3905,101,345,479,1802411,4193979,210,280048
,72,139,4,2,20,2,169,13,19,46,5,39,96,548,29,2,2,1,2,1,2,2,7,4,1,2,2,2,2,2,2,353,513,186,1,1,158,3,2,2,2,2,2,4,2,3,3,269,551,782,49,8,7,9,8,5,5,20,16,
3,811523',kBL:'4bzV'};google.sn='webhp';google.kHL='es-419';})();(function(){
var f=this||self;var h,k=[];function l(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||h}function m(a){for(var
function n(a,b,c,d,g){var e="";c||-1!==b.search("&ei=")||(e="&ei="+l(d),-1===b.search("&lei=")&&(d=m(d))&&(e+="&lei="+d));d="";!c&&f._cshid&&-1===b.se
+a+"&cad="+b+e+"&zx="+Date.now()+d;/^http:/i.test(c)&&"https:"===window.location.protocol&&(google.ml&&google.ml(Error("a"),!1,{src:c,glmm:1}),c="");r
oogle.log=function(a,b,c,d,g){if(c=n(a,b,c,d,g)){a=new Image;var e=k.length;k[e]=a;a.onerror=a.onload=a.onabort=function(){delete k[e]};a.src=c}};goog
google.y={};google.sy=[];google.x=function(a,b){if(a)var c=a.id;else{do c=Math.random();while(google.y[c])}google.y[c]=[a,b];return!1};google.sx=funct
e.lm,a)};google.lq=[];google.load=function(a,b,c){google.lq.push([[a],b,c])};google.loadAll=function(a,b){google.lq.push([a,b])};google.bx=!1;google.l
document.documentElement.addEventListener("submit",function(b){var a;if(a=b.target){var c=a.getAttribute("data-submitfalse");a="1"===c||"q"===c&&!a.el
.documentElement.addEventListener("click",function(b){var a;a:{for(a=b.target;a&&a!==document.documentElement;a=a.parentElement)if("A"===a.tagName){a=
);</script><style>#gbar,#guser{font-size:13px;padding-top:1px !important;}#gbar{height:22px}#guser{padding-bottom:7px !important;text-align:right}.gbh
x;width:100%}@media all{.gb1{height:22px;margin-right:.5em;vertical-align:top}#gbar{float:left}}a.gb1,a.gb4{text-decoration:underline !important}a.gb1
portant}
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:3px 8px 0}td{line-height:.8em}.gac_m td{line-h
l}.lst{height:25px;width:496px}.gsfi,.lst{font:18px arial,sans-serif}.gsfs{font:17px arial,sans-serif}.ds{display:inline-box;display:inline-block;marg
{color:#4b11a8;text-decoration:none}a:hover,a:active{text-decoration:underline}.fl a{color:#1558d6}a:visited{color:#4b11a8}.sblc{padding-top:5px}.sblc
```



```
▶ http -v GET https://httpbin.org/get
GET /get HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: httpbin.org
User-Agent: HTTPie/2.6.0



HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 299
Content-Type: application/json
Date: Mon, 15 Nov 2021 23:32:57 GMT
Server: gunicorn/19.9.0

{
    "args": {},
    "headers": {
        "Accept": "*/*",
        "Accept-Encoding": "gzip, deflate",
        "Host": "httpbin.org",
        "User-Agent": "HTTPie/2.6.0",
        "X-Amzn-Trace-Id": "Root=1-6192ee29-58b632df6e51d7592bdc95ae"
    },
    "origin": "201.105.252.203",
    "url": "https://httpbin.org/get"
}
```

Métodos HTTP comúnes

LIS

Licenciatura en Ingeniería de Software

Los métodos HTTP

- Informan al servidor que tipo de operación o interacción desea realizar el cliente.
- Las interacciones comunes incluyen:
    - recuperar un recurso
    - crear un nuevo recurso
    - realizar un cálculo
    - eliminar un recurso

- Los métodos HTTP comunes para sistemas en red son:

  - **GET**: Recupera un recurso del servidor, la respuesta puede almacenarse en caché.
  - **HEAD**: Solicita solo el encabezado de respuesta.
  - **POST**: Envía datos al servidor para almacenamiento o cálculos, la respuesta no se puede almacenar en caché.

- **PUT**: Envía datos al servidor, a menudo como un reemplazo de los datos existentes, la respuesta no se puede almacenar en caché.
- **PATCH**: Envía datos al servidor, a menudo como una actualización parcial de los datos existentes; la respuesta no se puede almacenar en caché.
- **DELETE**: Elimina un recurso existente en el servidor; la respuesta no se puede almacenar en caché.

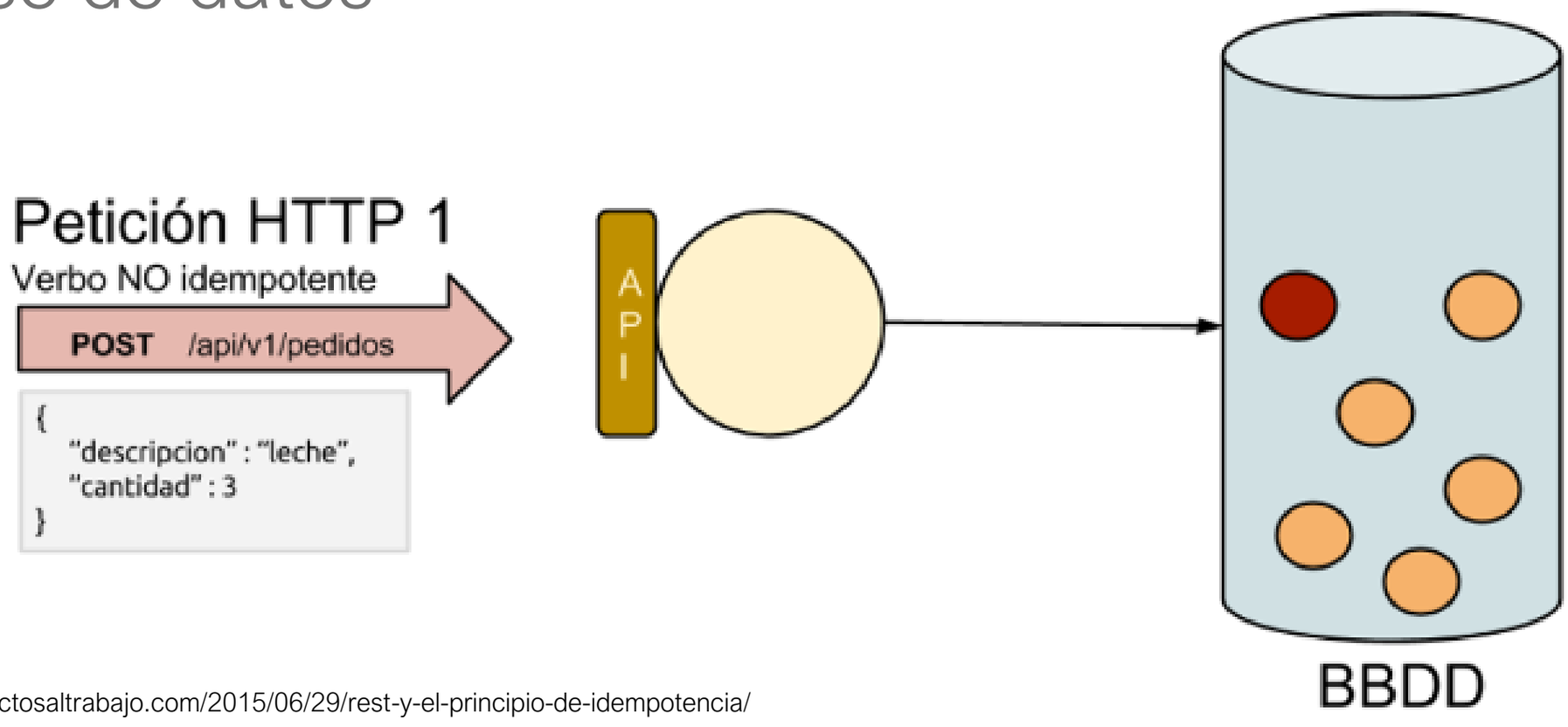
- **Safety** indica que el método HTTP utilizado no generará efectos secundarios, ej. alteración de datos
    - GET y HEAD recuperan recursos y no alteran datos.
- Si se implementan operaciones que alteran datos con métodos HTTP seguros se pueden generar resultados impredecibles

- Los métodos **idempotentes** aseguran que se produzcan los mismos efectos secundarios cuando se envían solicitudes idénticas.
- Cierto para **GET** y **HEAD** ya que no se modifican datos.
- La especificación HTTP garantiza que **PUT** y **DELETE** son idempotentes.
  - **PUT** reemplaza el recurso con una representación completamente nueva.
  - **DELETE** elimina el recurso del servidor.

- No se garantiza que **POST** sea idempotente
  - Pueden crear nuevos recursos en cada solicitud posterior; o
  - Alterar los datos de alguna manera que no se garantiza que produzca los mismos resultados, ej. Incrementando un valor.
- **PATCH** no es idempotente, solo se altera un subconjunto de campos no la representación completa.

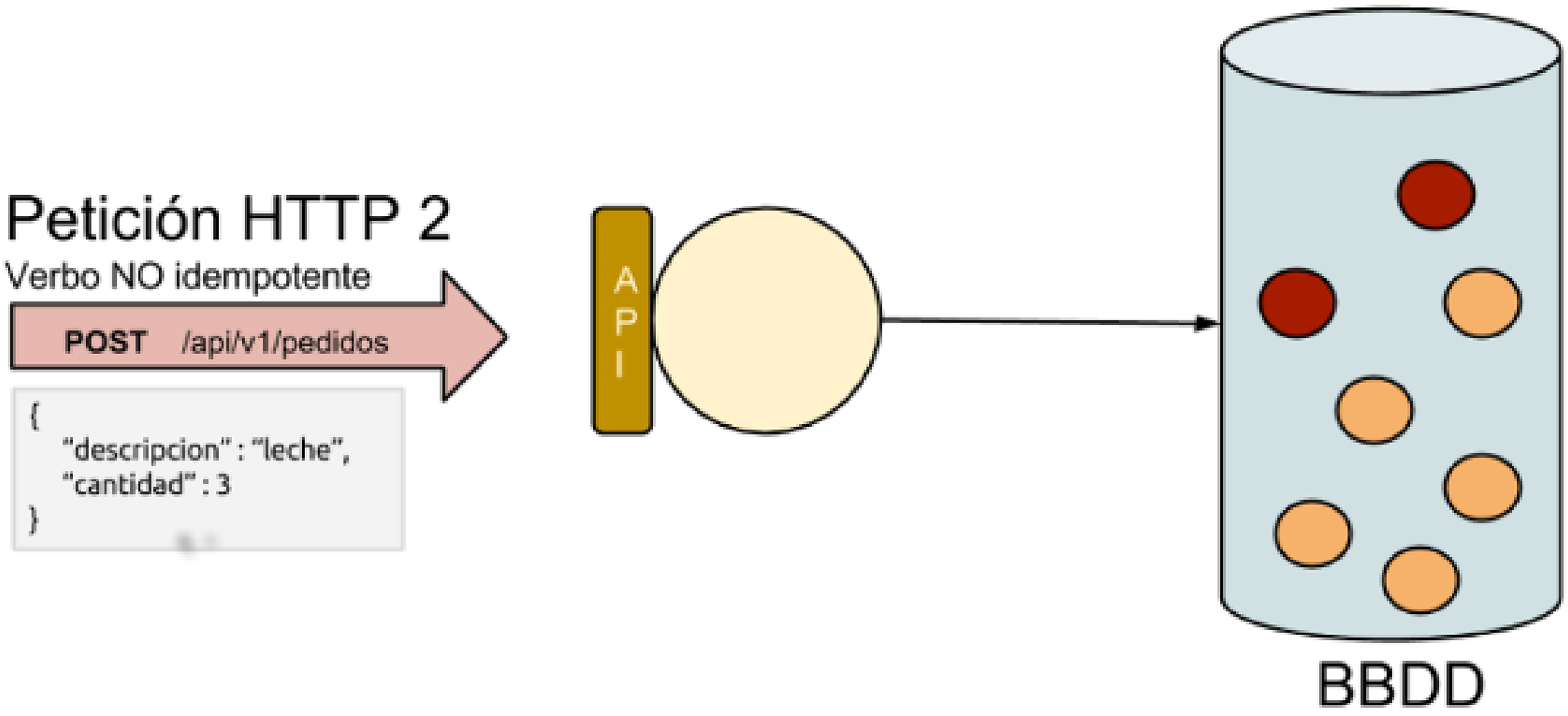
Imaginemos la ejecución repetida de una petición utilizando un **verbo NO idempotente** con los mismos datos de entrada sobre un mismo recurso.

Paso 1: Enviamos una petición de creación de un recurso con unos parámetros. Y el estado en el sistema se modifica, concretamente se crea un nuevo pedido en base de datos



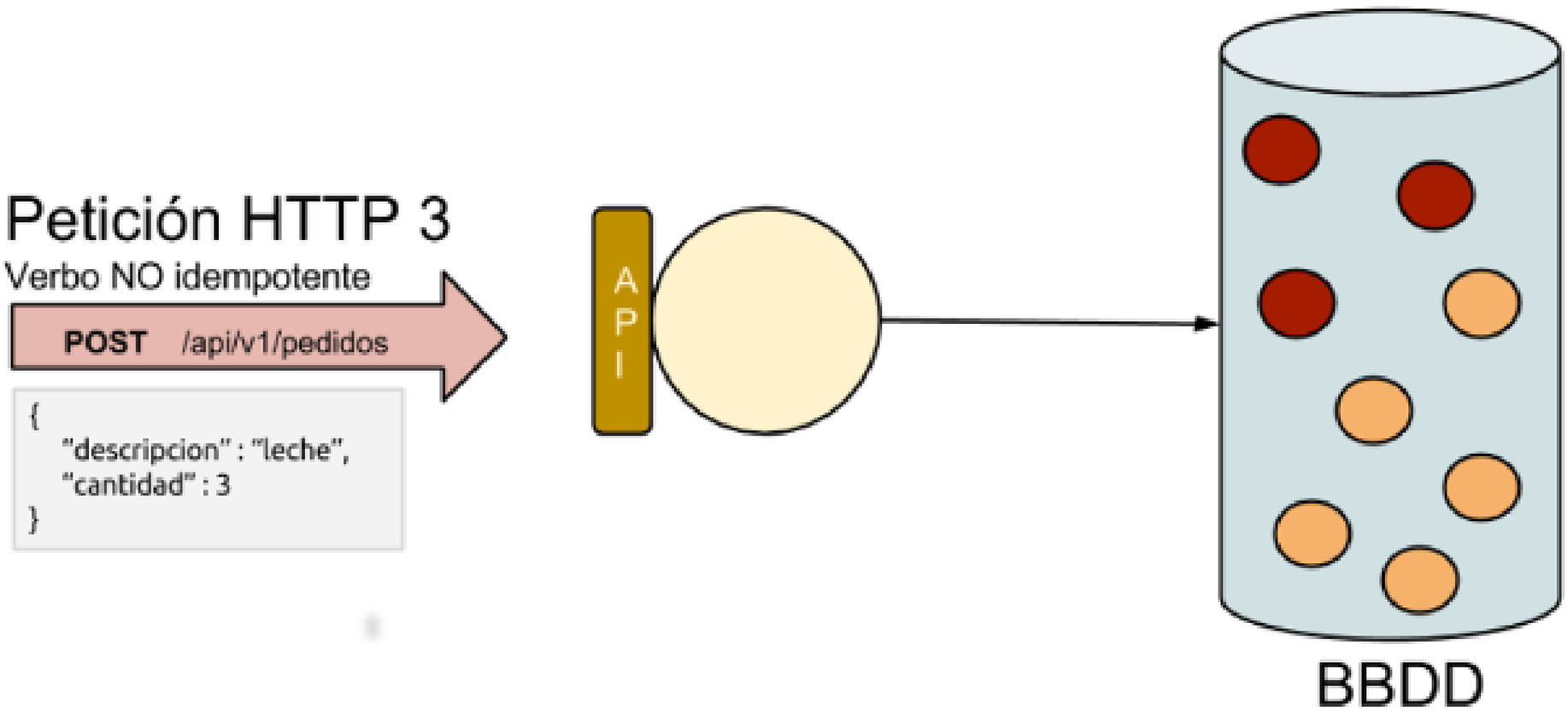
https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

Paso 2: Volvemos a enviar la misma petición de creación al mismo recurso con los mismos parámetros. Observamos que el estado del sistema vuelve a cambiar, se crea otro recurso nuevo.



https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

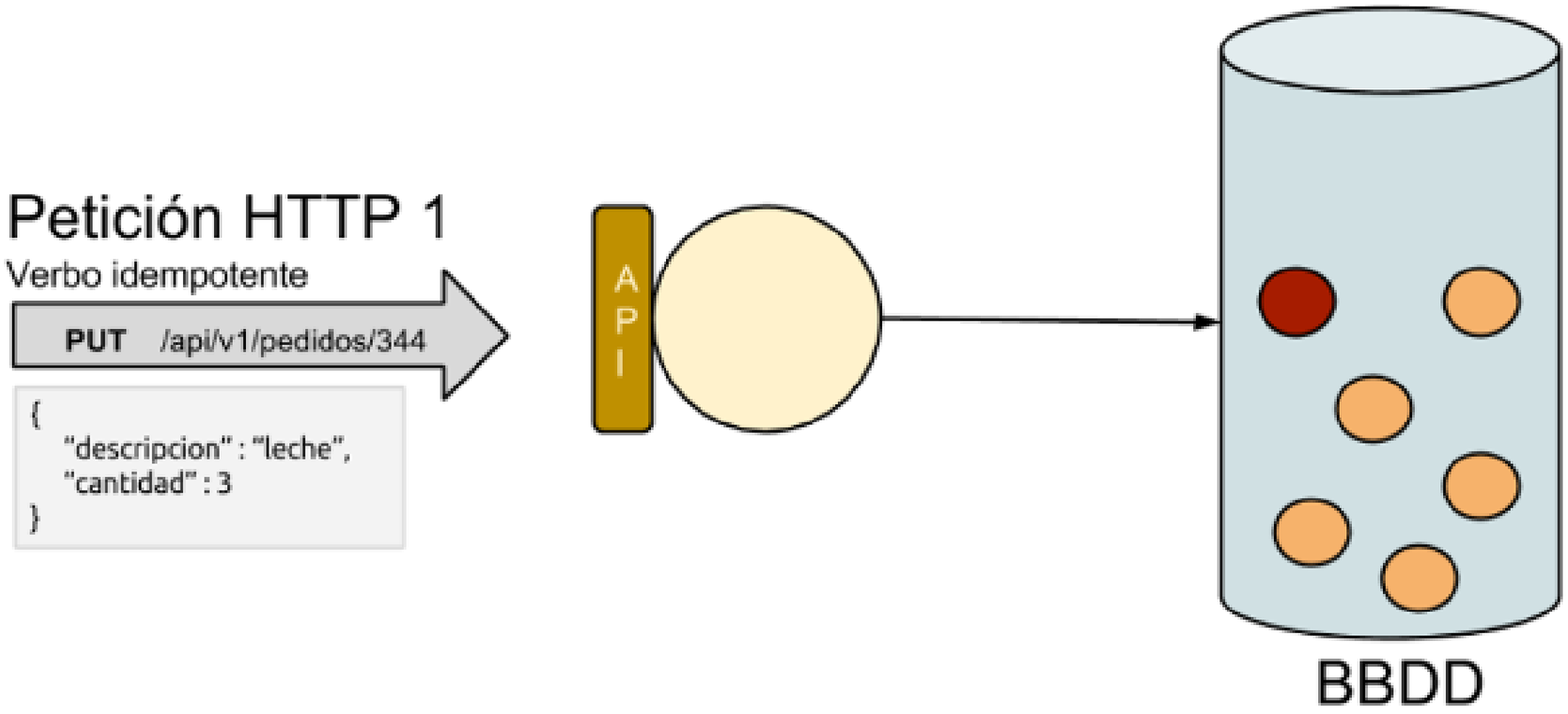
Paso 3: Nuevamente enviamos la misma petición de creación al mismo recurso con los mismos parámetros. Y se a crear otro recurso nuevo.



https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

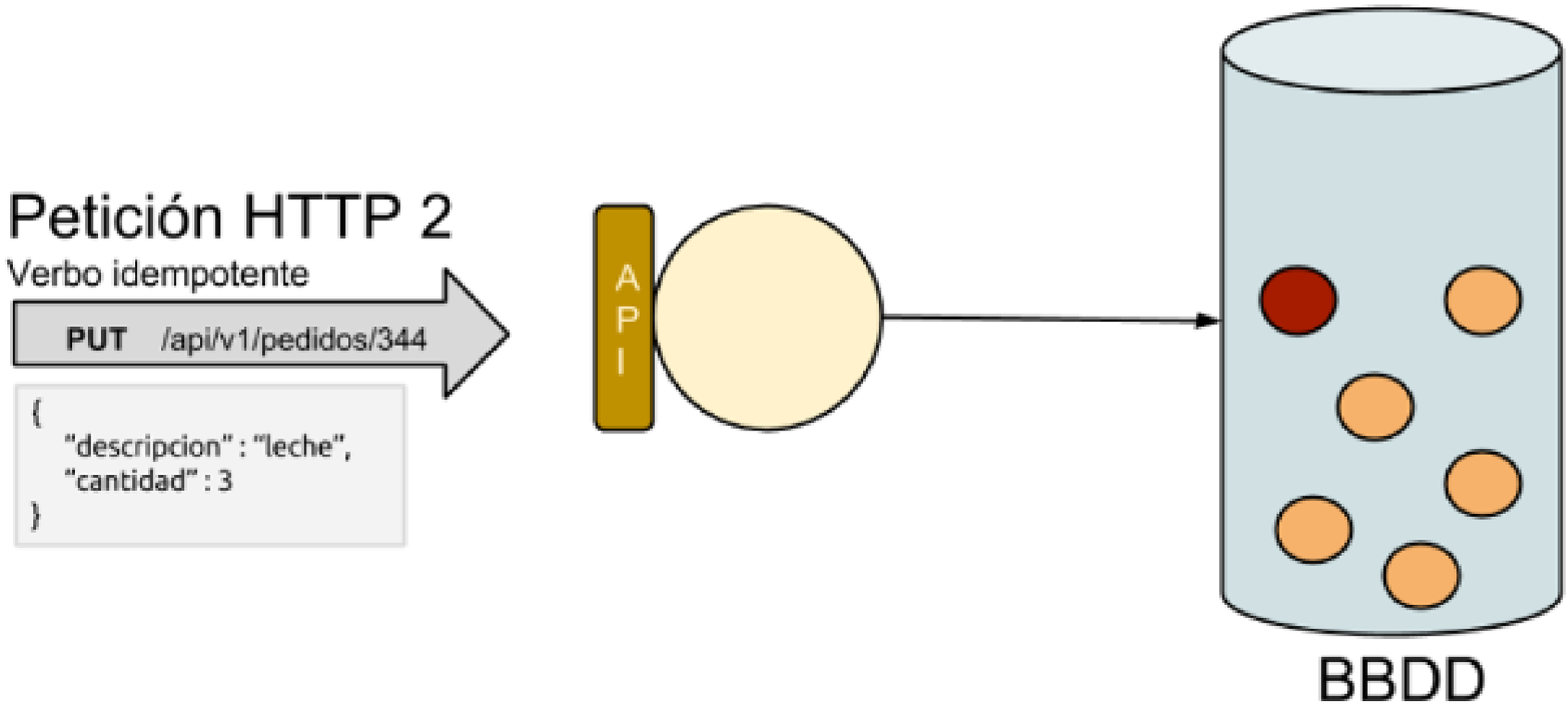
Paso 1: Veamos el comportamiento esperado de un **verbo idempotente**.

Enviamos una petición de creación/actualización sobre un recurso con unos determinados parámetros. Y se crea un nuevo recurso en el sistema en caso de no existir o se modifica, en caso de que algo haya cambiado, el existente.



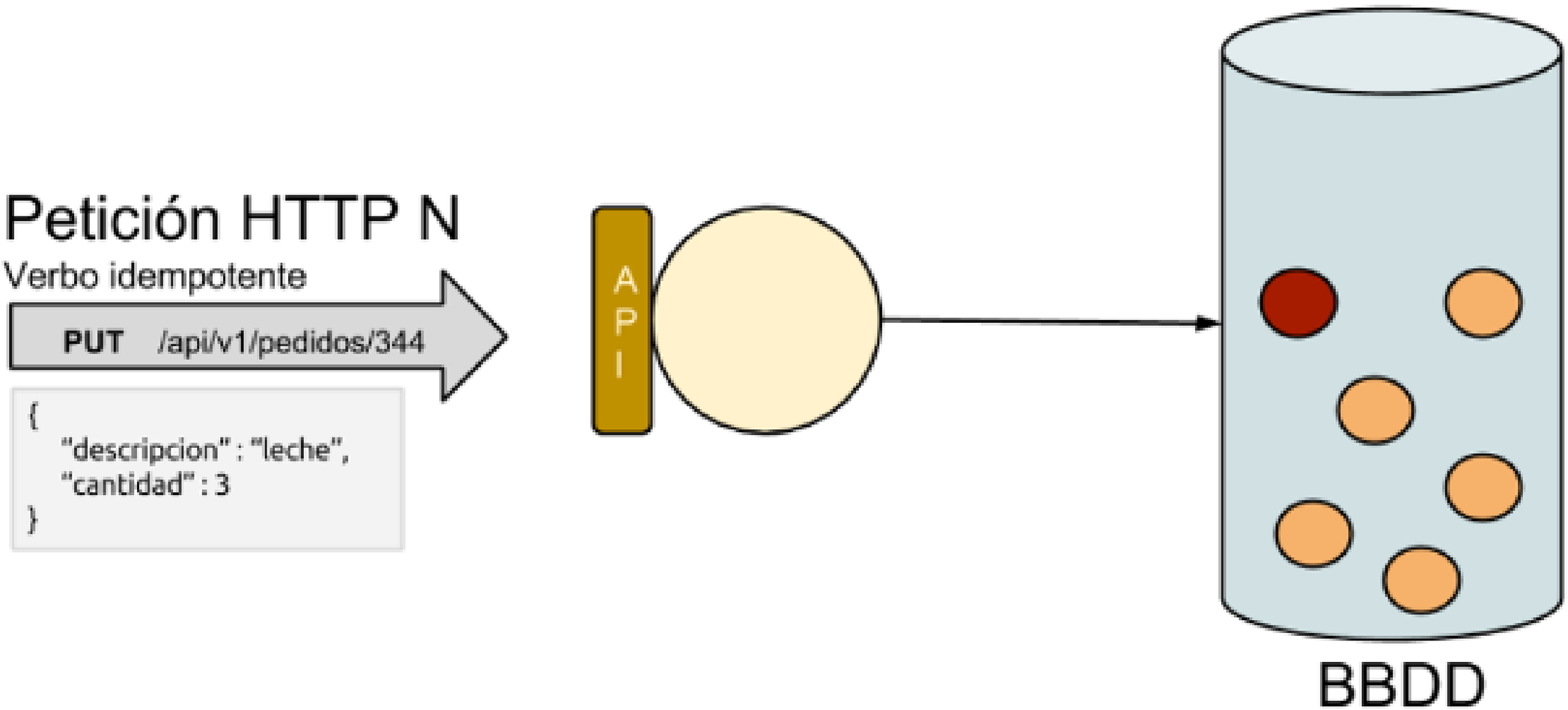
https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

Paso 2: Repetimos la petición con el mismo verbo idempotente con los mismos parámetros y sobre el mismo recurso.



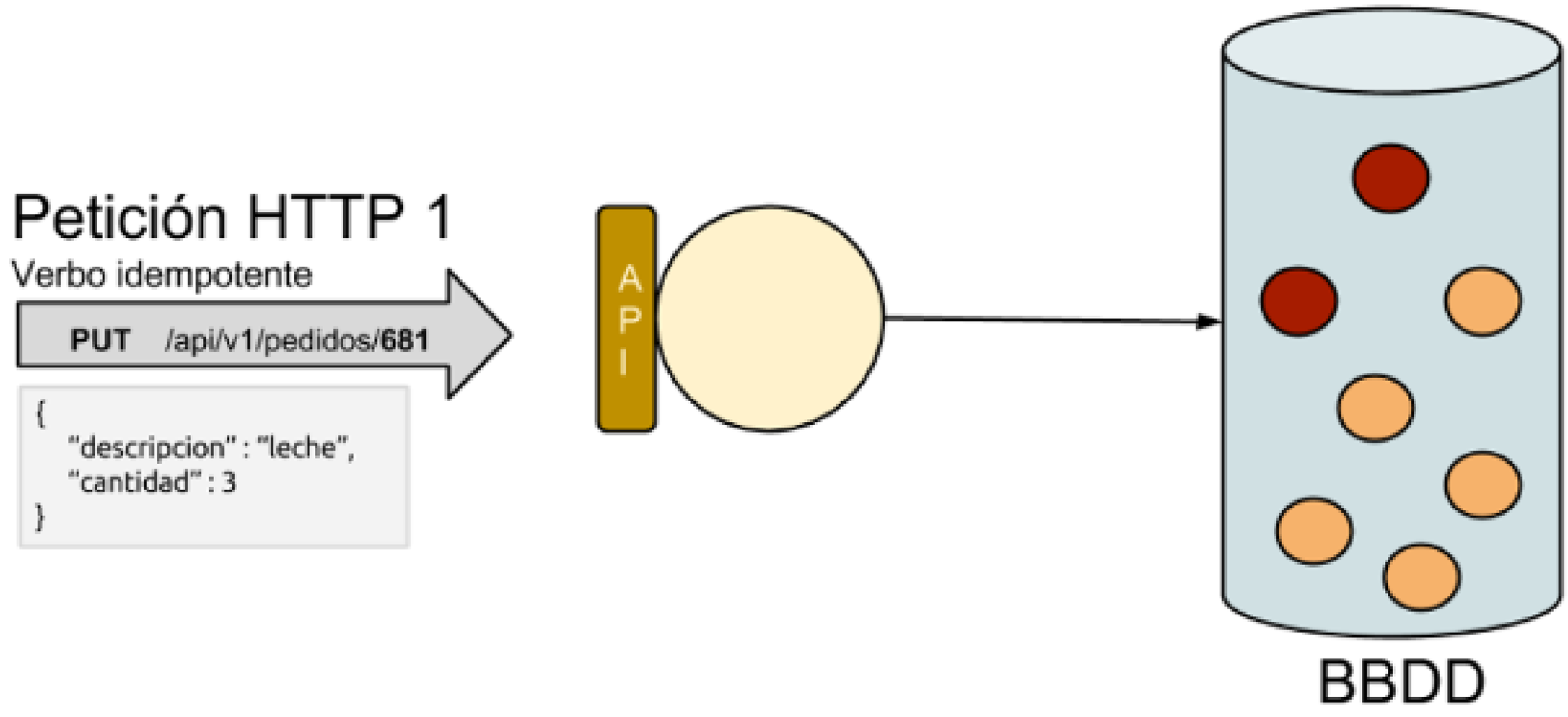
https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

Paso 3: Repetimos N veces la petición y el estado sigue siendo el mismo.



https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

Paso 4: Si cambiamos algo, por ejemplo el recurso al que atacamos utilizando los mismos parámetros y mismo verbo idempotente se producirían cambios en el sistema, ya que la petición es contra un nuevo recurso.



https://www.adictosaltrabajo.com/2015/06/29/rest-y-el-principio-de-idempotencia/

Método/Verbo	Seguro	Idempotente
GET	SÍ	SÍ
POST	No	No
PUT	No	SÍ
PATCH	No	No
DELETE	No	SÍ
HEAD	SÍ	SÍ
OPTIONS	SÍ	SÍ

Códigos de respuesta

- Las respuestas HTTP incluyen un código de respuesta que indican al consumidor si la solicitud tuvo éxito o no.
- Existen cinco grupos, los principales son:
  - 200 Éxito: Indican que la solicitud de se proceso con éxito.
  - 300 Redirecciones: Indican que el cliente puede necesitar tomar acciones adicionales para completar la solicitud, como seguir un redireccionamiento.

- 400 Errores cliente: indican una falla en la solicitud que el cliente podría querer corregir y volver a enviar.

- 500 Errores servidor: indican una falla en el servidor que no es culpa del cliente. El cliente puede volver a intentarlo en el futuro, si corresponde.

- La lista completa se puede consultar en:

  - https://developer.mozilla.org/es/docs/Web/HTTP/Status

- Los códigos que son comunes en el diseño de sistemas en red son los siguientes:

Código	Descripción
200 OK	La solicitud ha tenido éxito.
201 Created	La solicitud se ha cumplido y ha dado lugar a la creación de un nuevo recurso.
202 Accepted	La solicitud ha sido aceptada para su procesamiento, pero el procesamiento no se ha completado.
204 No content	El servidor ha cumplido la solicitud pero no necesita devolver un *cuerpo*. Esto es común para las operaciones de eliminación.
304 Not Modified	El servidor determinó que el contenido no ha cambiado desde la última solicitud según lo determinado por el encabezado de solicitud If-Modified-Since o If-None-Match proporcionado por el cliente.

Código	Descripción
400 Bad Request	El servidor no pudo entender la solicitud debido a una sintaxis incorrecta.
401 Unauthorized	La solicitud requiere autenticación del usuario.
403 Forbidden	El servidor entendió la solicitud, pero se niega a cumplirla.
404 Not found	El servidor no ha encontrado nada que coincida con el URL/URI solicitado.
412 Precondition failed	El cliente envió una solicitud con una condición basada en la última marca de tiempo modificada o eTag y la condición falló. El cliente debe recuperar el recurso e intentar el cambio nuevamente, si lo desea.

Código	Descripción
415 Unsupported Media Type	El servidor no pudo responder con ninguno de los tipos de medios soportados de acuerdo con lo especificado en el encabezado Accept.
428 Precondition Required	El servidor requiere que se proporcione un encabezado de condición previa antes de que se pueda procesar la solicitud. A menudo se aplica cuando se requieren encabezados de control de concurrencia.
500 Internal Server Error	El servidor encontró una condición inesperada que le impidió cumplir con la solicitud.

Negociación de contenido

- Permite a los clientes solicitar uno o más tipos de medios.
- Una operación puede admitir diferentes representaciones de recursos, como CSV, PDF, PNG, JPG, SVG, etc.
- La solicitud es mediante el encabezado Aceptar. Ej:

```
GET https://api.example.com/projects HTTP/1.1
Accept: application/json
```

- Se puede incluir más de un tipo de medio compatible en el encabezado, como se muestra en este ejemplo:

```
GET https://api.example.com/projects HTTP/1.1
Accept: application/json,application/xml
```

- Se puede utilizar * como comodín al seleccionar tipos de medios.
- `text/*` indica que se acepta cualquier subtipo del tipo de medio de texto.

- `*/*` indica que el cliente aceptará cualquier tipo de medio en la respuesta.
  - Común para navegadores, preguntarán al usuario si desea guardar el archivo o iniciar una aplicación.
  - Los clientes de una API deben ser explícitos para evitar errores de tiempo de ejecución que podrían ocurrir al encontrar un tipo de medio desconocido o no compatible

Factores de calidad

- También conocidos como `quality values`, `q-values` o `q-factors`

- Permiten especificar la preferencia por tipos de medios específicos en el encabezado `Acept`.

- Se expresan como un valor `q` entre `0` y `1` para asignar un orden preferido.

- El servidor revisa los valores del encabezado y devuelve la respuesta utilizando el tipo de contenido que coincide con lo que él admite y con lo que solicitó el cliente.

- Si el servidor no puede responder con un tipo de contenido solicitado, devuelve un código de respuesta `415 Unsupported Media Type`.

- Ejemplo del uso de `qvalues` para especificar una preferencia por XML, con soporte para JSON si XML no está disponible:

```
GET https://api.example.com/projects HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0
```

- Respuesta al ejemplo anterior:

```
HTTP/1.1 200 OK
Date: Fri, 22 November 2019 06:57:43 GMT
Content-Type: application/xml

<project>...</project>
```

- La negociación de contenido amplía el soporte de tipos de medios de una API más allá de JSON o XML.

- Permite que algunas o todas las operaciones de una API respondan con el tipo de contenido que mejor se adapte a las necesidades del cliente de la API.

- La negociación de idiomas permite que las API admitan varios idiomas en una respuesta.

  - Similar a la negociación de contenido

  - Se utilizan el encabezado de solicitud Accept-Language y el encabezado de respuesta Content-Language.

Control de Caché

- **La solicitud de red más rápida es la que no es necesario realizar.**

- Almacenamiento local de datos para evitar la recuperación de los datos en el futuro, optimizando las comunicaciones de red.

- Existen herramientas de almacenamiento en caché del lado del servidor, como Memcached

    - Mantiene los datos en la memoria y reduce la necesidad de obtener datos sin cambios de una base de datos

    - Mejora el rendimiento de la aplicación.

- El control de caché HTTP permite que los clientes API o los servidores de caché intermediarios almacenen localmente las respuestas.
    - Acerca la memoria caché al cliente; y
    - Reduce o elimina la necesidad de atravesar la red hasta llegar al servidor.
        - Los usuarios experimentan un mejor rendimiento y una menor dependencia de la red.
- Existen varias opciones de almacenamiento en caché a través del encabezado de respuesta `Cache-Control`.
    - Declara si la respuesta se puede almacenar en caché y durante cuánto tiempo se debe almacenar.

- Aquí hay una **respuesta** de ejemplo de una operación API que **devuelve una lista de proyectos**:

```
HTTP/1.1 200 OK
Date: Tue, 22 December 2020 06:57:43 GMT
Content-Type: application/xml
Cache-Control: max-age=240

<project>...</project>
```

- `max-age` indica que los datos pueden almacenarse en caché hasta 240 segundos antes de que el cliente los considere obsoletos.

- Las API también pueden marcar explícitamente una respuesta como no almacenable en caché.
    - Se necesita una nueva solicitud cada vez que se requiera la respuesta.

```
HTTP/1.1 200 OK
Date: Tue, 22 December 2020 06:57:43 GMT
Content-Type: application/xml
Cache-Control: no-cache

<project>...</project>
```

- La aplicación cuidadosa del control de caché reduce el tráfico de red y acelera las aplicaciones.

Solicitudes condicionales

- Permiten a los clientes solicitar una representación de los recursos actualizada **solo <u>si</u>** algo ha cambiado.

- Los clientes que envíen una solicitud condicional recibirán:
  - 304 Not Modified si el contenido no ha cambiado, o
  - 200 OK junto con el contenido que cambió

- Hay dos tipos de precondiciones para informar al servidor sobre la copia en caché local del cliente a comparar:
  - `time-based` y `entity tag-based`

- Las precondiciones `time-based` requieren que el cliente almacene el encabezado de respuesta `Last-Modified`
- El encabezado de solicitud `If-Modified-Since` se usa para especificar la última marca de tiempo modificada.
- El servidor la usará para comparar con la última marca de tiempo modificada conocida.
- Así se determina si el recurso ha cambiado.

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0

HTTP/1.1 200 OK
Date: Tue, 22 December 2020 06:57:43 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
Last-Modified: Tue, 22 December 2020 05:29:03 GMT

<project>...</project>
```

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0
If-Modified-Since: Tue, 22 December 2020 05:29:03 GMT

HTTP/1.1 304 Not Modified
Date: Tue, 22 December 2020 07:03:43 GMT
```

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0
If-Modified-Since: Tue, 22 December 2020 07:33:03 GMT

Date: Tue, 22 December 2020 07:33:04 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
Last-Modified: Tue, 22 December 2020 07:12:01 GMT

<project>...</project>
```

- El `entity tag` o `ETag` es un valor que representa el estado actual de recurso.

- El cliente puede almacenar la `ETag` después de una solicitud GET, POST o PUT, utilizando el valor para verificar los cambios a través de una solicitud HEAD o GET

- Una `ETag` es *hash* de toda la respuesta.

- Alternativamente, los servidores pueden proporcionar una `weak ETag`.

  - Semánticamente equivalente pero quizás no equivalente byte por byte.

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0

HTTP/1.1 200 OK
Date: Tue, 22 December 2020 06:57:43 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
ETag: “16f0ffa99ed5aae4edffdd6596d7131f”

<project>...</project>
```

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0
If-None-Match: “16f0ffa99ed5aae4edffdd6596d7131f”

HTTP/1.1 304 Not Modified
Date: Tue, 22 December 2020 07:03:43 GMT
```

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0
If-None-Match: “16f0ffa99ed5aae4edffdd6596d7131f”

HTTP/1.1 200 OK
Date: Tue, 22 December 2020 07:33:04 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
ETag: “a128d66ac3ec050b4fd2c2a6264fbaf51db10fab”


<project>...</project>
```

- Las solicitudes condicionales reducen el esfuerzo necesario para validar y recuperar los recursos almacenados en caché.
- Las ETag son valores que representan el estado interno actual.
- Las marcas de tiempo de última modificación se usa para la comparación basada en el tiempo.
    - También pueden utilizarse para el control de concurrencia al realizar modificaciones en los recursos.

Control de concurrencia en HTTP

- La concurrencia es un desafío cuando se desarrollan APIs que modifican datos por diferentes usuarios al mismo tiempo.

- HTTP tiene incorporado un control de concurrencia.
    - Las solicitudes condicionales se utilizan para este fin.

- Combinar `ETags` o `timestamps` de última modificación con métodos de cambio de estado como PUT, PATCH o DELETE, puede asegurar que otro cliente no sobrescriba los datos a través otra solicitud.

- Para aplicar una solicitud condicional:
    - El cliente agrega una precondición para evitar la modificación si la marca de tiempo de última modificación o `ETag` del recurso han cambiado.
    - Si la precondición no se cumple, el servidor envía una respuesta `412 Precondition Failed`.
- Los servidores API también pueden imponer el requisito de un encabezado de precondición.
    - `428 Precondition Required` si no se encontró ninguno de los encabezados condicionales en la solicitud.

- Ejemplo en el que dos clientes intentan modificar un proyecto:
  - Primero, cada cliente recupera el recurso del proyecto mediante una solicitud GET.
  - Luego cada uno intenta un cambio, pero solo el primer cliente puede aplicar el cambio.

```
GET /projects/12345 HTTP/1.1
Accept: application/json;q=0.5,application/xml;q=1.0

HTTP/1.1 200 OK
Date: Tue, 22 December 2020 07:33:04 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
ETag: “a252d76ab3fc050b4fd2c3a6363fdaf52db10ecb”

<project>...</project>
```

```
PUT /projects/1234
If-Match: “a252d76ab3fc050b4fd2c3a6363fdaf52db10ecb”

{ “name”:"Proyecto 1234", “Description”:"Mi proyecto" }

HTTP/1.1 200 OK
Date: Tue, 22 December 2020 08:21:20 GMT
Content-Type: application/xml
Cache-Control: max-age=240
Location: /projects/12345
ETag: “1d7209c9d54e1a9c4cf730be411eff1424ff2fb6”

<project>...</project>
```

```
PUT /projects/1234
If-Match: “a252d76ab3fc050b4fd2c3a6363fdaf52db10ecb”

{ “name”: “Proyecto 5678”, “Description”: “No, mi proyecto” }

HTTP/1.1 412 Precondition Failed
Date: Tue, 22 December 2020 08:21:24 GMT
```

- El segundo cliente que recibió la respuesta de Precondition Failed ahora debe:
  - Recuperar la representación actual de la instancia del recurso
  - Informar al usuario de los cambios; y
  - Preguntar si el usuario desea volver a enviar los cambios realizados o dejarlo como está

- El control de concurrencia se puede agregar a una API a través de precondiciones HTTP en el encabezado de la solicitud.
  - Si la fecha de última modificación/ETag no ha cambiado, la solicitud se procesa normalmente.
  - Si ha cambiado, se devuelve un código de respuesta 412 y se evita que el cliente sobrescriba los datos como resultado de que dos clientes independientes modifiquen el mismo recurso al mismo tiempo.
- Esta poderosa capacidad integrada en HTTP evita que los equipos tengan que inventar su propio soporte de control de concurrencia.

Conclusiones

- HTTP es un protocolo versatil con un conjunto sólido de funcionalidades, incluidas algunas que son menos conocidas.
- El uso de la negociación de contenido permite que los clientes y servidores de una API acuerden un tipo de contenido.
- Las directivas de control de caché brindan soporte de almacenamiento en caché intermediario y del lado del cliente.

- Las precondiciones de HTTP se pueden usar para:
  - Determinar si las memorias caché caducadas siguen siendo válidas
  - Proteger los recursos para que no se sobrescriban cambios.
- Al aplicar estas técnicas, los equipos pueden crear API robustas que impulsan aplicaciones complejas de manera resilente y evolutiva.