Aplicaciones y Servicios en Redes

Tema 04. Aplicaciones y Servicios Móviles

Alberto Eloy García Gutiérrez

Luis Sánchez González

DPTO. DE INGENIERÍA DE COMUNICACIONES

Contenido

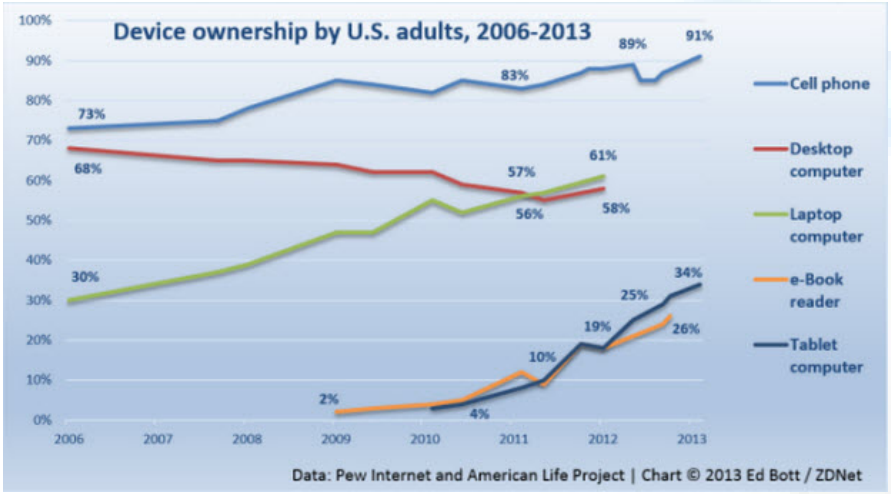
- Introducción
- Servicios Web ligeros. REST
- Servicios de mensajería

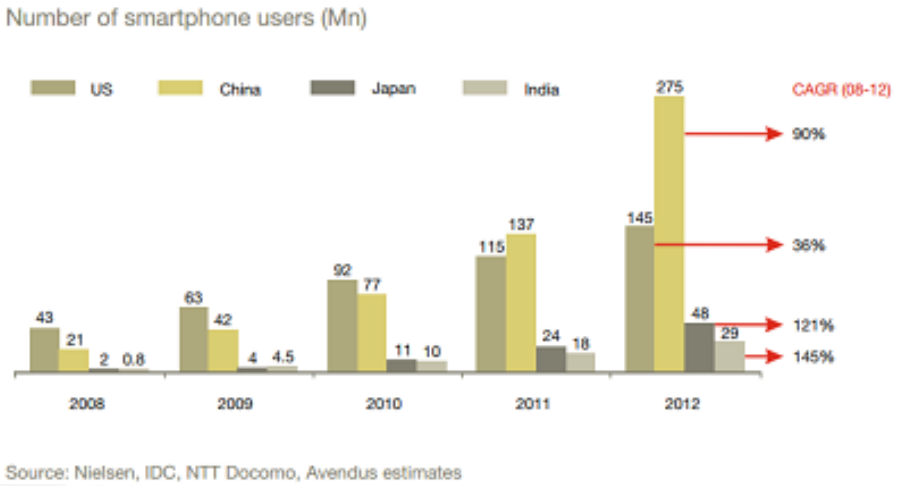
Contenido

- Introducción
  - Motivación
  - Retos de la prestación de servicios en movilidad
  - Qué vamos a tratar – Qué no vamos a tratar
- Servicios Web ligeros. REST
- Servicios de mensajería

Aplicaciones y servicios móviles. Motivación

- El mundo se hace móvil





Retos de la prestación de servicios en movilidad

- Debidos a la movilidad en sí
  - Localizar al dispositivo móvil o al servicio
  - Soportar las desconexiones
  - Hibernar de manera eficiente
  - Reconectarse rápidamente
- Debidos a los dispositivos
  - Capacidad de cómputo
  - Batería
- Debidos a las condiciones de contorno
  - Variabilidad en el contexto

Retos de la prestación de servicios en movilidad

- La movilidad ya no sólo está en el nivel de red

Movilidad del cliente
- El dispositivo se mueve entre puntos de acceso
- El dispositivo se mueve entre redes de acceso

Movilidad de la identidad
- Acceso al servicio desde diferentes dispositivos
- Acceso al servicio a través de diferentes perfiles

Movilidad del contenido
- Distribución del contenido en función del usuario
- Enrutamiento del contenido para soportar grandes anchos de banda

Movilidad del servicio
- Modelo de Servicios en la nube
- Flexibilidad en la prestación del servicio

Movilidad de la aplicación
- Computación en la nube
- Modelos Software-as-a-Service

Qué vamos a tratar – Qué no vamos a tratar

- Si veremos
  - Evolución del paradigma web service
  - Servicios de mensajería como habilitadores de las comunicaciones M2M
- No veremos
  - Desarrollo de aplicaciones móviles
  - Paradigma del Cloud computing

Contenido

- Introducción
- Servicios Web ligeros. REST
  - Introducción
  - REST
  - REST vs SOAP
  - JSON vs XML
- Servicios de mensajería

REST – Introducción

- SOAP (Simple Object Access Protocol)
  - No es orientado a objetos
  - No es simple
- Con el pretexto de que es sencillo y está basado en la web
  - lleva equívocamente a centrarse en el protocolo y los mensajes
  - y no en la abstracción
  - (cuestiones ya superadas incluso en RPC)
- WS-*
  - Conjunto de protocolos relacionados con los servicios Web
  - Decenas de especificaciones
  - No implementadas (o parcialmente) muchas de ellas

REST – Introducción

- SOAP trata de llevar el concepto de Web
  - Realmente es hacer un túnel sobre HTTP en XML
  - En realidad usa HTTP como transporte y no sigue los principios de la Web
- ¿Por qué surge REST?
  - WS-* es muy completo pero muy complejo

REST – Introducción

- Servicio de agenda:
  - Usando WS-*:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.example.com/agenda">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

  - Usando REST:

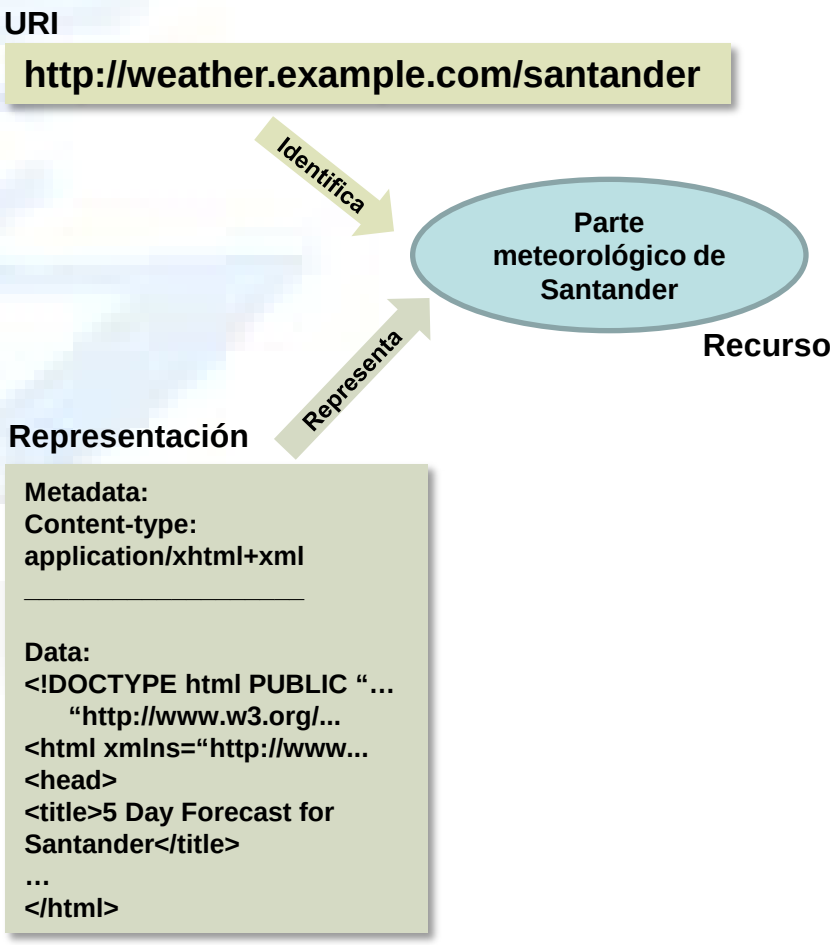
```
http://www.example.com/agenda/UserDetails/12345
```

REST – Introducción

- REST (*Representational State Transfer*) es una técnica de desarrollo de software web que se sustenta sobre el protocolo HTTP.
- Es un estilo de arquitectura, no es una especificación ni un estándar
- Surge de la tesis de doctorado de Roy Fielding
  - Define restricciones para que un sistema sea REST
  - Se basa en la Web estática
- Cliente – Servidor

REST – Fundamentos

- Recursos
  - Cualquier cosa que pueda identificar usando una URI
- URI
  - Dirección universal a un recurso
- Representación
  - Lo que se envía al cliente cuando solicita un recurso



REST – Fundamentos

- Principios o Restricciones
  - Recursos deben ser uniformemente accesibles (URI única)
  - Recursos son accedidos y actualizados por operaciones GET, POST, PUT, DELETE
  - Metadatos para describir recursos
  - Comunicación entre cliente y servidor debe ser "*stateless*"
    - No depender del estado del recurso, sino de la URI que representan
    - El servidor no guarda estado sobre la conversación con el cliente. El cliente mantiene el estado y provee en la petición la información necesaria

http://www.bing.com/search?q=universidad+cantabria&qs=n&filt=all&**first=1**
vs
http://www.bing.com/search?q=universidad+cantabria&qs=n&filt=all&**first=100**

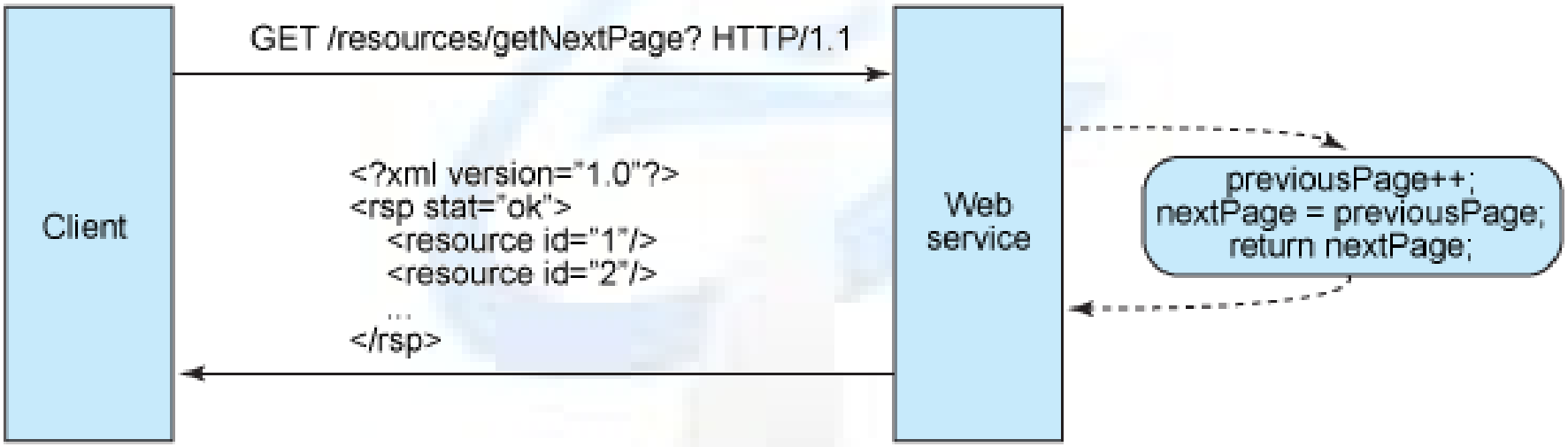
Podemos acceder al resultado 100 sin pasar por los anteriores

REST – Fundamentos

- En REST hay estado, pero sólo en los recursos, no en la aplicación (o sesión).
- Una petición del cliente al servidor debe contener TODA la información necesaria para entender la misma.
- No puede basarse en información previa asociada a la comunicación (por ejemplo, información de sesión).
- No puede haber datos de sesiones del cliente en el servidor
  - El servidor sólo guarda y maneja el estado de los recursos que aloja.
- Es el cliente el que debe guardar su estado (y enviárselo al servidor).
- Esto permite escalabilidad (servidores más sencillos).
  - Implica mayor ancho de banda, pero facilita las caches

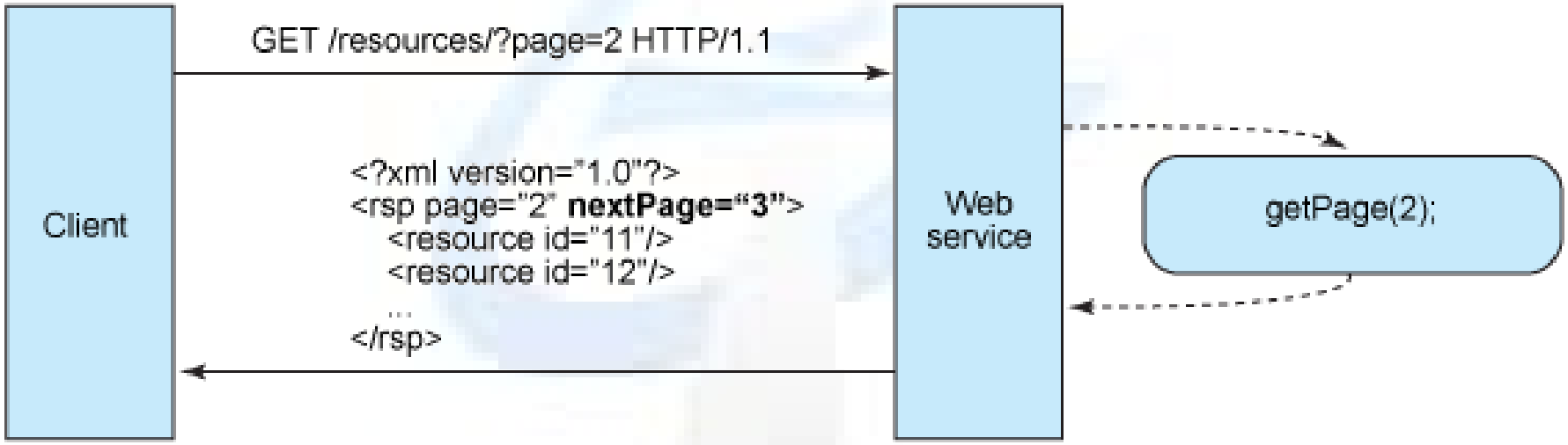
REST – Fundamentos

- Servicio “stateful”



REST – Fundamentos

- Servicio “stateless”



REST – Recursos

- Dos tipos de recursos:
  - Colecciones – http://example.com/resources/
  - Elementos – http://example.com/resources/142
  - Los métodos tienen un significado ligeramente diferente se trate de una colección o un elemento

REST – URI

- Describen los recursos
  - Las URI son únicas y deben ser descriptivas.
  - Al construir una URI, ésta debe ser semántica
    - Usando las convenciones establecidas:
      - "/" : para expresar herencia, por ejemplo: /europa/españa/cantabria
      - " , " : para atributos no ordenados, por ejemplo: /europa/españa/madrid,asturias,murcia
      - ";" : para atributos ordenados, por ejemplo: /europa/españa/barcelona/lat=30.3;lng=4.11
      - "?": para modificadores, por ejemplo: /europa/españa?search="Huelva"

REST – URI

- El recurso se identifica en la petición HTTP

```
GET /sqlrest/CUSTOMER/1234 HTTP/1.1
Host: www.thomas-bayer.com
Accept: text/html, text/xml
```

URI = http://www.thomas-bayer.com/sqlrest/CUSTOMER/1234

- Paso de parámetros al servicio

```
http://www.example.com/agenda/UserDetails?firstName=John&lastName=Doe
```

REST – Operaciones

- Funcionamiento
  - Orientado a recursos, no a operaciones
  - Una invocación REST es una petición HTTP
  - Las operaciones sobre recursos son limitadas
    - CRUD

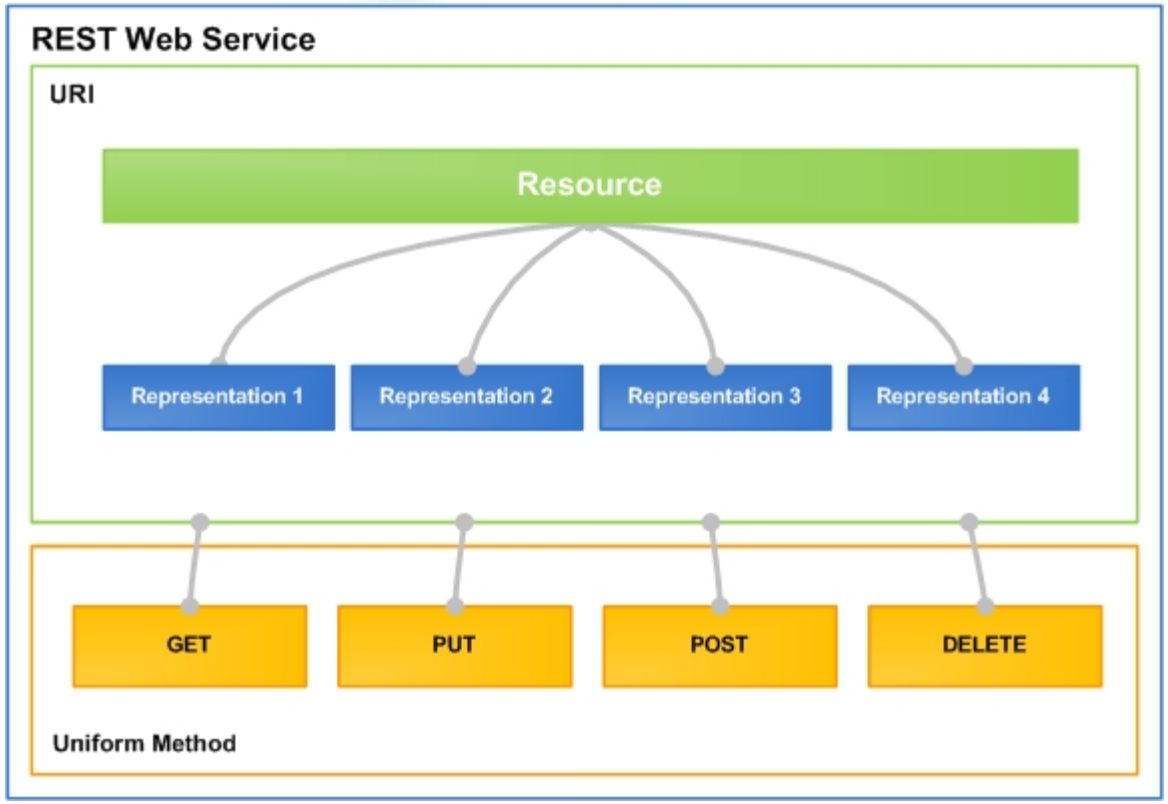
Operación	Método HTTP
Create	POST
Retrieve	GET
Update	PUT
Delete	DELETE

REST – Operaciones

- Operaciones seguras y no seguras
  - Sólo GET es una operación segura ya que es la única que no modifica el recurso
  - PUT, POST y DELETE son operaciones no seguras ya que pueden modificar un recurso
- Operaciones idempotentes y no idempotentes
  - GET, PUT y DELETE son operaciones idempotentes ya que el resultado de la operación no cambiará independientemente del número de veces que se realice la operación

REST – Representaciones

- Un recurso tiene múltiples representaciones



REST – Representaciones

- Las representaciones muestran el estado actual del recurso en un formato dado

```
GET /profile/1234
Host: example.com
Accept: text/html

GET /profile/1234
Host: example.com
Accept: text/x-vcard

GET /profile/1234
Host: example.com
Accept: application/mi-vocabulario-propio+xml
```

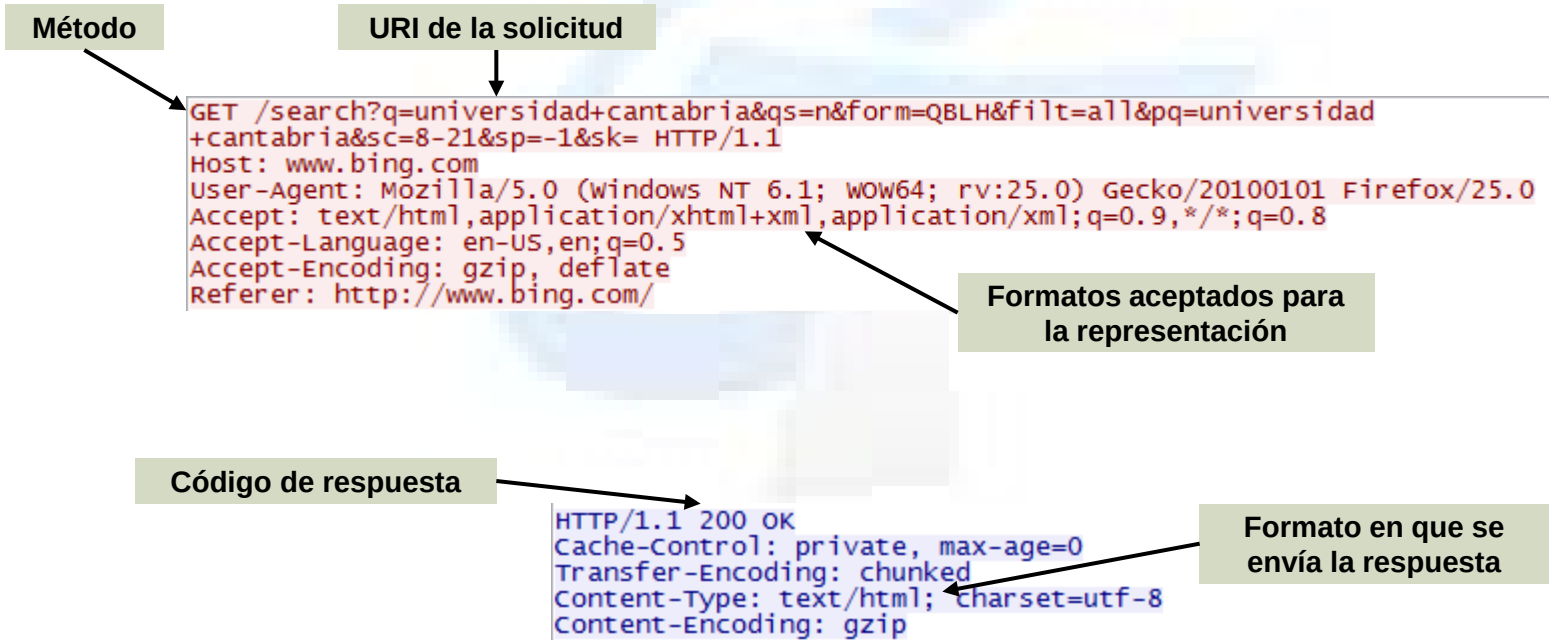
REST – Representaciones

- Cada representación también puede tener una funcionalidad diferente

URI	GET	PUT	POST	DELETE
http://www.ejemplo.com/usuarios/	Listar Usuarios	-	Crear usuario	-
http://www.ejemplo.com/usuarios/user1	Mostrar user1	Crear user1	Modificar user1	Borrar user1
http://www.ejemplo.com/usuarios/user1/mapa	Mostrar Mapa user1	Crear Mapa user1	-	Borrar Mapa user1

REST – Uso

- Ejemplo de invocación REST



REST – Ejemplos

- http://www.thomas-bayer.com/sqlrest/

```
GET /sqlrest/ HTTP/1.1
Host: www.thomas-bayer.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Date: Sat, 07 Dec 2013 12:45:28 GMT
Content-Length: 466

<?xml version="1.0"?><resource xmlns:xlink="http://www.w3.org/1999/xlink">
    <CUSTOMERList xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/">CUSTOMER</
CUSTOMERList>
    <INVOICEList xlink:href="http://www.thomas-bayer.com/sqlrest/INVOICE/">INVOICE</
INVOICEList>
    <ITEMList xlink:href="http://www.thomas-bayer.com/sqlrest/ITEM/">ITEM</ITEMList>
    <PRODUCTList xlink:href="http://www.thomas-bayer.com/sqlrest/PRODUCT/">PRODUCT</
PRODUCTList>
</resource>
```

REST – Ejemplos

- Twitter REST API
  - https://dev.twitter.com/docs/api/1.1

Request

```
GET /1.1/statuses/show/123.json?trim_user=1
HTTP/1.1
X-HostCommonName: api.twitter.com
Host: api.twitter.com
X-Target-URI: https://api.twitter.com
Connection: Keep-Alive
```

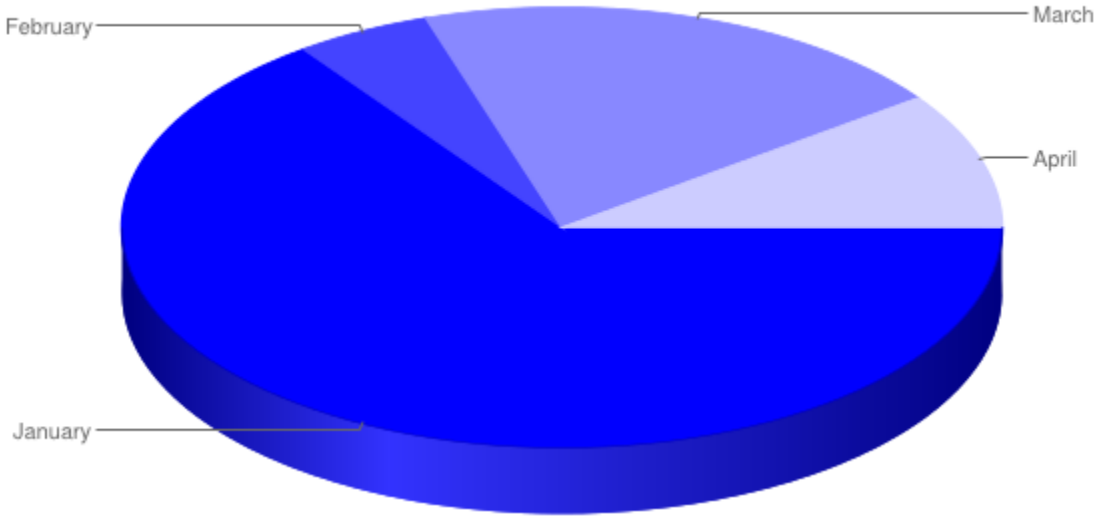
Response Snapshot

```
HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
set-cookie: guest_id=v1%3A138642248420983676;
Domain=.twitter.com; Path=/; Expires=Mon, 07-Dec-2015
13:21:24 UTC
content-length: 61
server: tfe
strict-transport-security: max-age=631138519
date: Sat, 07 Dec 2013 13:21:24 UTC
Connection: close

{
  "errors": [
    {
      "message": "Bad Authentication data",
      "code": 215
    }
  ]
}
```

REST – Ejemplos

- http://chart.apis.google.com/chart?cht=p3&chd=t:65,5,20,10&chco=0000FF&chs=900x300&chl=January|February|March|April



REST – Ejemplos

- http://chart.apis.google.com/chart?cht=p3&chd=t:65,5,20,10&chco=0000FF&chs=900x300&chl=January|February|March|April

```
GET /chart?cht=p3&chd=t:65,5,20,10&chco=0000FF&chs=900x300&chl=January|February|March|April
HTTP/1.1
Host: chart.apis.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:26.0) Gecko/20100101 Firefox/26.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie:
PREF=ID=e57f41809c24a2cb:U=510c3348d2f24225:FF=0:LD=en:CR=2:TM=1334052768:LM=1385744340:GM=1:S=
3t1mnvHgpobUZ87j;
NID=67=YB4u_Jh7ndjp_G_4rZ6Fsne6WXr6zbkicOLONvlaWIkPOXRKsc6FDxweEzmvn8zbUWfpem_KmtTrcvDp7nGGKWRj
YpJ1jSYOjgWjjBQvCoZTUn4gtogvcMUmU1z3dJ3ErOpQ4XSl7Kud_Y-jyJfQMt49YO7cw8FxXD8NnVpJCpZyxQYQ-
QwvuDw-haHuOs53NRIG_Hd1wXJ-mu7U5QYZOUtWN8YiYtu5xZHUjmJ3FjiURBciAjISKhDOJQ; SID=DQAAAM8AAAArRI-
WI3cY5s4ptVJmSXmZUrpKa9Uau1svepG3HqtUfgozcfUZnB4NBlBK2yt-
hzfYTYnRf5_VmAzLqDhpcQk3ZLYHjddcCVqCN_GhMIgsGc9ds-9F2jkC1YIgsRz_7NkCrVG7drbjhSkbcpG9iB3bKshd6O6
FqR3UNe2wvP3TF49UOPLIEAopGMe9SDoD3RJbR_LkmpOHKUvrcD__VQnsmmsUt1tQi-PO8ZCSBFKjAMMoekCVs-
Ntx3YCGwaSr6BWE_X2YkDpXp7Me_UBWSn_; HSID=AdsnMbFk4HHf-HRgf; APISID=V7H2GzlupjHfgCl7/
Aub6KKzCJcMsUPXN4
Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 16 Dec 2013 08:21:50 GMT
Expires: Tue, 17 Dec 2013 08:21:50 GMT
Cache-Control: public, max-age=86400
Last-Modified: Mon, 09-Sep-2013 18:04:30 GMT
X-Frame-Options: ALLOWALL
Access-Control-Allow-Origin: *
Content-Type: image/png
X-Content-Type-Options: nosniff
Server: GoogleChartAPI/1.0
Content-Length: 16833
X-XSS-Protection: 1; mode=block

.PNG
```

REST – Ventajas

- Maximiza la reutilización
  - Todos los recursos tienen identificadores = hacen más grande la Web
  - La visibilidad de los recursos que forman una aplicación/servicio permite usos no previstos
- Minimiza el acoplamiento y permite la evolución
  - La interfaz uniforme esconde detalles de implementación
  - El uso de hipertexto permite que sólo la primera URL usada para acceder acople un sistema a otro
- Elimina condiciones de fallo parcial
  - Fallo del servidor no afecta al cliente
  - El estado siempre puede ser recuperado como un recurso
- Escalado sin límites
  - Los servicios pueden ser replicados en cluster, cacheados y ayudados por sistemas intermediarios

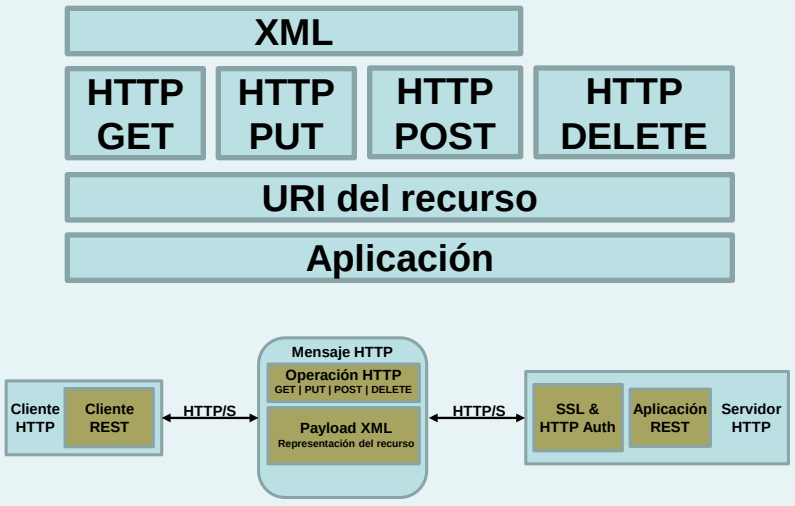
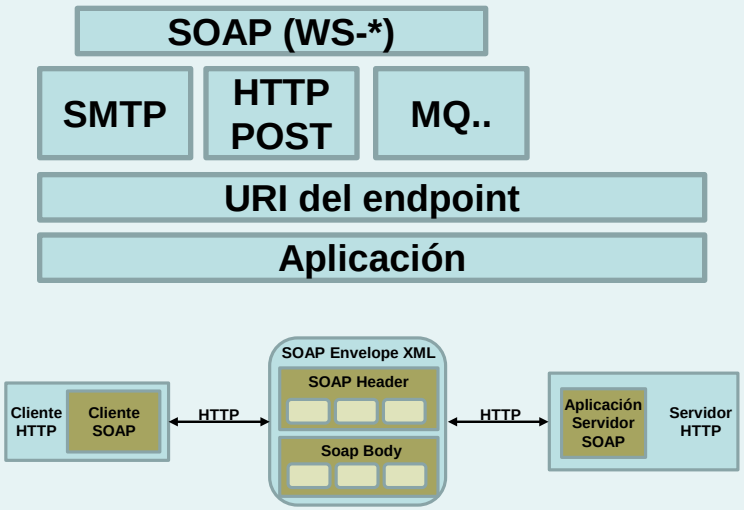
REST vs WS-*

	REST	WS-*
Concepto	**"La Web es el universo de la información accesible globalmente" (Tim Berners Lee)**	**"La Web es el transporte universal de mensajes"**
Características	Las operaciones se definen en los mensajes. Una dirección única para cada instancia del proceso. **Cada objeto soporta las operaciones estándares definidas.** **Componentes débilmente acoplados.**	**Las operaciones son definidas como puertos WSDL.** **Dirección única para todas las operaciones.** Múltiple instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.

REST vs WS-*

	REST	WS-*
Ventajas declaradas	**Bajo consumo de recursos.** Las instancias del proceso son creadas explícitamente. **El cliente no necesita información de enrutamiento a partir de la URI inicial.** Los clientes pueden tener una interfaz "listener" (escuchadora) genérica para las notificaciones. Generalmente fácil de construir y adoptar.	Fácil (generalmente) de utilizar. La depuración es posible. Las operaciones complejas pueden ser escondidas detrás de una fachada. **Envolver APIs existentes es sencillo** Incrementa la privacidad. Herramientas de desarrollo.
Posibles desventajas	Gran número de objetos. **Manejar el espacio de nombres (URIs) puede ser engorroso.** **La descripción sintáctica/semántica muy informal (orientada al usuario).** Pocas herramientas de desarrollo.	**Los clientes necesitan saber las operaciones y su semántica antes del uso.** Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. Las instancias del proceso son creadas implícitamente.

REST vs WS-*

	REST	WS-*
Protocolo		
	XML autodescriptivo.	XML Schema
	HTTP	Independiente del transporte
	HTTP es un protocolo de aplicación	HTTP es un protocolo de transporte

REST vs WS-*

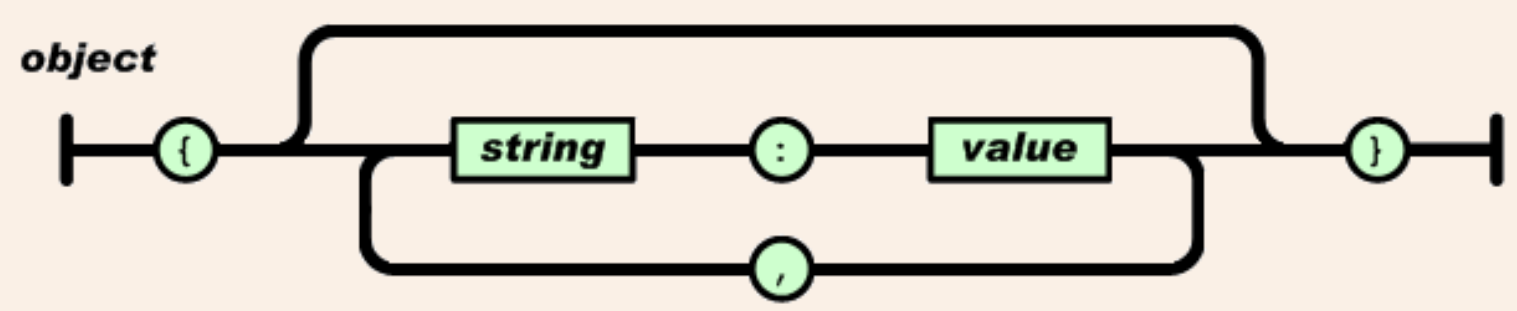
	REST	WS-*
Descripción del servicio	Confía en documentos orientados al usuario que define las direcciones de petición y las respuestas	WSDL
	Interactuar con el servicio supone horas de testeado y depuración de URIs.	Se pueden construir automáticamente stubs (clientes) por medio del WSDL
Gestión del estado	El servidor no tiene estado (stateless).	El servidor puede mantener el estado de la conversación.
	Los clientes mantienen el estado siguiendo los enlaces.	Los clientes mantienen el estado suponiendo el estado del servicio.
Seguridad	HTTPS	WS-Security

REST – JSON

- JSON (JavaScript Object Notation)
  - Formato de intercambio de información ligero
  - Fácil de generar y parsear
  - Auto-descriptivo y fácil de entender
  - Independiente del lenguaje de programación
  - Capaz de reproducir jerarquías en los datos
  - Tipo MIME: application/json

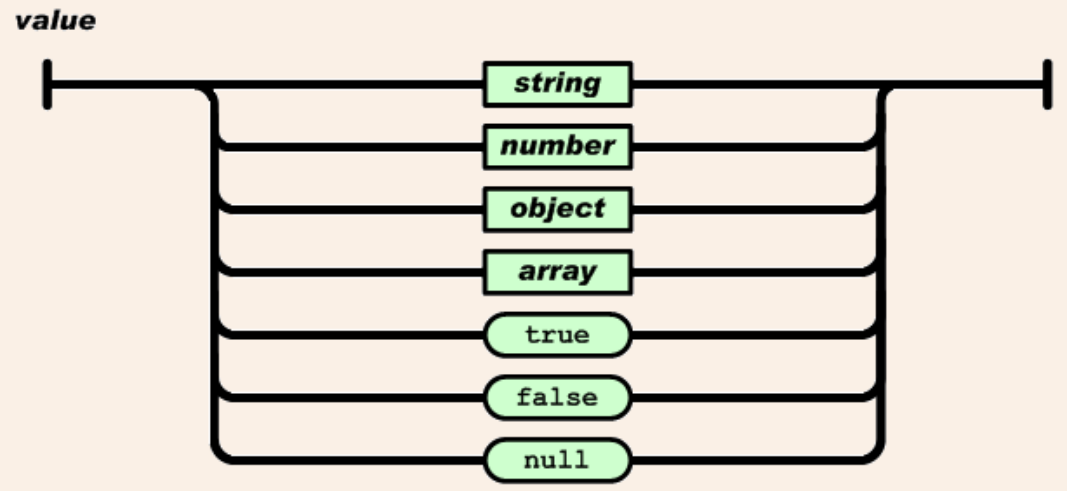
REST – JSON

- Un objeto JSON es una colección de duplas nombre/valor
  - Los objetos se definen entre los signos { y }
  - Los miembros de un objeto son duplas nombre/valor donde nombre y valor se separan por el signo :
  - Las distintas duplas de un objeto se separan por el signo ,



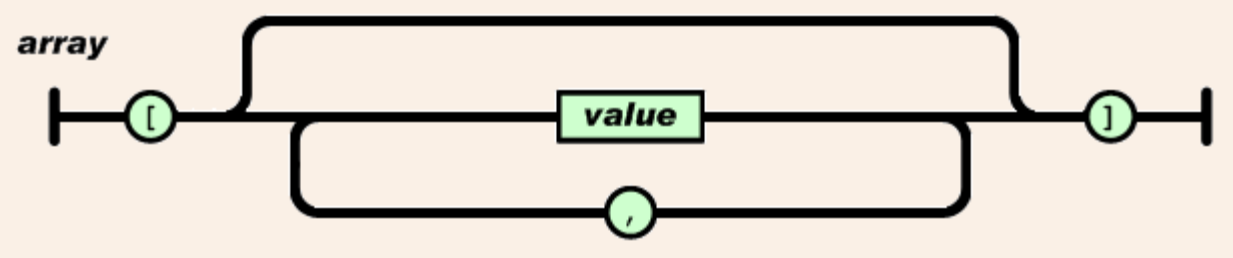
REST – JSON

- El campo valor dentro de una dupla puede ser:
  - Un string
  - Un número
  - Un objeto
  - Un array
  - Un boolean (TRUE o FALSE)
  - Puede estar vacío



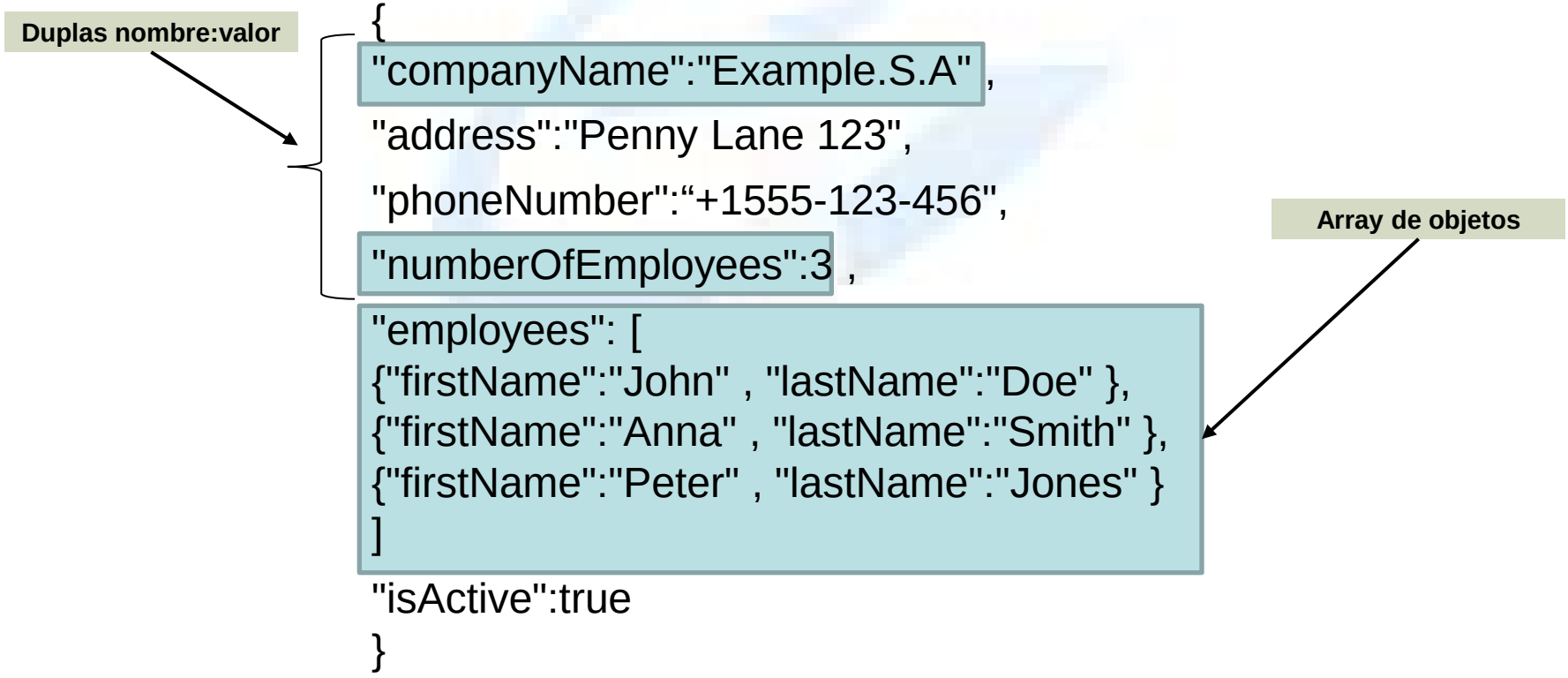
REST – JSON

- Los array son conjuntos de valores
  - Se definen entre los signos [y]
  - Los valores se separan por el signo ,



REST – JSON

- Ejemplo de documento JSON



JSON vs XML

- Semejanzas
  - Basados en texto plano
  - Auto-descriptivos (orientados al humano)
  - Soporta jerarquías (valores dentro de valores)
  - Fácil de parsear
- Diferencias
  - Sin etiquetas de final
  - Más corto
  - Más rápido de leer y escribir
  - Usa arrays
  - No hay palabras reservadas

Contenido

- Introducción
- Servicios Web ligeros. REST
- Servicios de mensajería
  - Introducción
  - MQTT

Internet de las Cosas

- Billones de cosas conectadas entre sí
- Aplicaciones variadas
  - http://www.youtube.com/watch?v=s9nrm8q5eGg
- Necesidades variadas
  - Heterogeneidad
  - Control
    - Gestión
    - Capacidades de las cosas
    - Estado de los dispositivos
  - Modelos de comunicación
  - Propiedades de la comunicación
    - Seguridad, QoS, Robustez

Internet de las Cosas

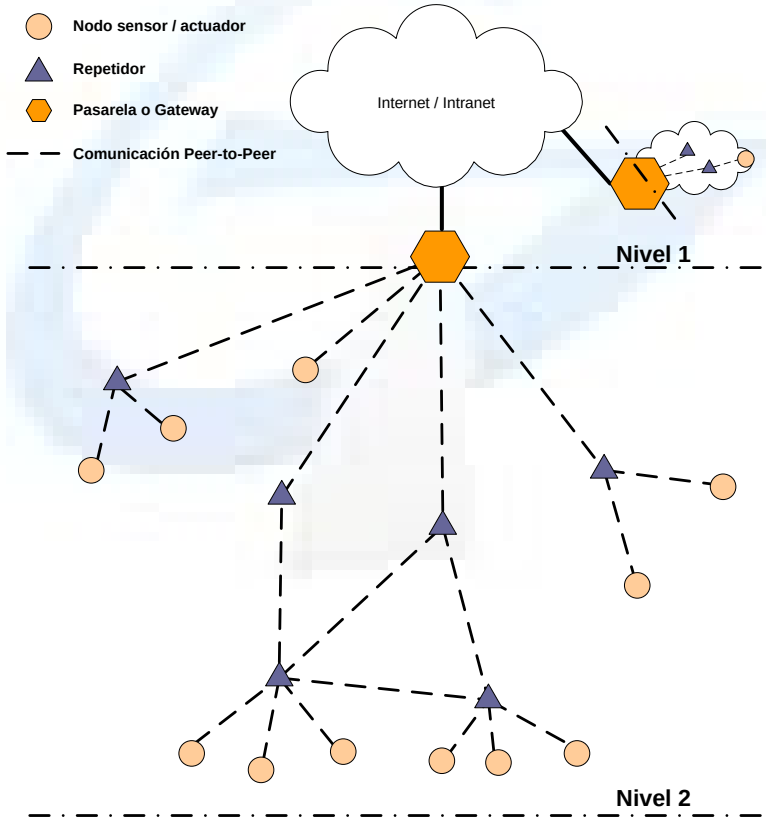
- ¿Qué hay de malo en los protocolos de la Internet actual?
  - HTTP revolucionó la forma en la que las personas consumen información
  - Tiene un único modelo sencillo pero válido: Mando una petición y Leo una respuesta
  - Disponible en múltitud de dispositivos hoy en día
- Los servicios disponibles en la Internet de las Cosas tienen retos distintos
  - Las fuentes de información no son conocidas
  - Modelo basado en eventos:
    - Uno a muchos
    - Escuchar permanentemente
    - Volúmenes de información enormes
    - A través de redes no fiables

Internet de las Cosas

- Otros retos a tener en cuenta:
  - Volumen = Coste
  - Consumo de batería
  - Pseudo tiempo real
  - Seguridad
  - Fiabilidad
  - Escalabilidad
- No hay una solución Extremo a Extremo
  - Divide y vencerás
  - Clusterizar la Internet de las Cosas
    - No muy diferente a las subredes de Internet

Internet de las Cosas

- Arquitectura de red jerárquica



Prestación de servicios orientada a eventos

- Otros protocolos que cubren esta parte del problema
  - MQTT
  - XMPP
  - AMQP
  - CoAP

MQTT

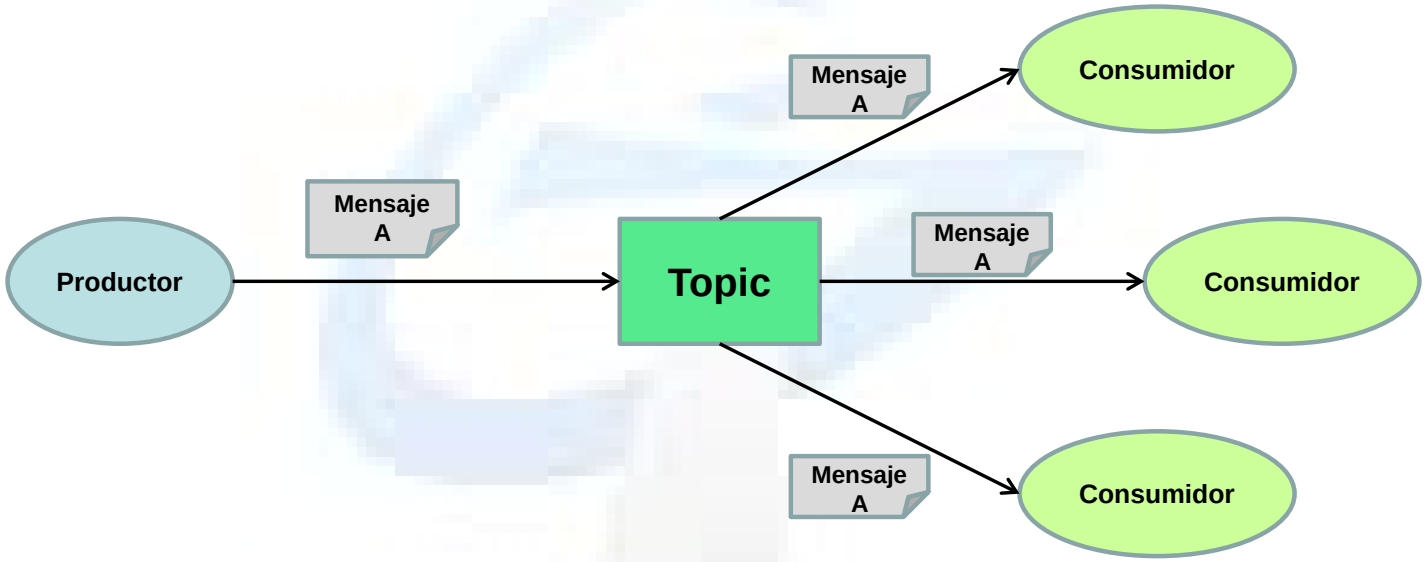
- Message Queing Telemetry Transport (MQTT)
  - Heredero del modelo de comunicación del intercambio de mensajes
  - Resumen de sus autores:
    - "A light weight event and message oriented protocol allowing devices to asynchronously communicate efficiently across constrained networks to remote systems"
  - Basado en un broker que hace de intermediario entre los productores de mensajes y sus consumidores

MQTT

- Criterios de diseño
  - Conectar fácilmente el mundo físico (M2M) al mundo IT
  - Soportar redes poco fiables (bajo ancho de banda, alta latencia, no fiable, caras (coste por byte)
  - Recursos a nivel computacional limitados (controladores de 8 bits, 256Kb RAM)
  - Acoplamiento ligero para soportar entornos dinámicos donde grandes cantidades de mensajes han de intercambiarse de formas no previstas de manera anticipada.
  - Permitir determinadas calidades en la entrega de mensajes fruto de los compromisos entre ancho de banda, disponibilidad y garantía de entrega
  - Capaz de soportar grandes cantidades de dispositivos (10K clientes MQTT)
  - Simple para los desarrolladores
  - Protocolo abierto

MQTT

- Protocolo basado en el modelo Publish – Subscribe



- Permite que un mensaje publicado por un solo productor sea recibido por múltiples consumidores
- El productor envía (publica) un mensaje (publicación) a un asunto (topic).
- Los consumidores reciben todos los mensajes que llegan a los asuntos a los que están subscritos
- Un servidor se encarga de asociar publicaciones con subscripciones
  - Si no hay asociaciones el mensaje se descarta
  - Si las hay, el mensaje se envía a los consumidores correspondientes

MQTT – Topics

- Forman el espacio de nombres
  - Es jerárquico con cada “sub-topic” separado por una “/”
  - Por ejemplo:
    - Una casa publica información sobre sí misma
      - <país>/<región>/<ciudad>/<código_postal>/<casa>/consumoElectricidad
      - <país>/<región>/<ciudad>/<código_postal>/<casa>/estadoAlarma
    - Y se subscribe a los comandos de control
      - <país>/<región>/<ciudad>/<código_postal>/<casa>/activarCalefaccion
  - Se pueden usar wildcards
    - Para un nivel se usa “+”
    - Para múltiples niveles se usa “#”
    - Por ejemplo:
      - ES/Cantabria/Santander/+/+/consumoElectricidad
      - ES/Cantabria/Santander/39005/#

MQTT – Subscripciones y Publicaciones

- Pueden ser permanentes (durable) o volátiles (non-durable)
  - Permanentes
    - Si el consumidor está conectado los mensajes se envían inmediatamente.
    - Si el consumidor no está conectado los mensajes se guardan en el servidor hasta que el consumidor se vuelve a conectar.
  - Volátiles
    - La subscripción expira si el consumidor se desconecta del servidor.
- Un mensaje puede ser retenido (retained)
  - Un productor puede marcar un mensaje como retained
  - El broker guarda el último mensaje retained que llega a cualquier topic
  - Los nuevos consumidores subscritos a un topic reciben inmediatamente el último mensaje retenido

MQTT – Uso eficiente de recursos

- Protocolo mínimo
  - El mensaje más pequeño mide 2 bytes
- Basado en el paradigma “push”
  - Cliente -> Servidor y Servidor -> Cliente
- Soporta interrupciones en la conexión
  - El servidor tiene persistencia
- Soporta el concepto de sesión
  - Mediante mensajes de keep-alive se sabe si el consumidor se ha desconectado
- Testeado sobre múltiples redes (vsat, GPRS, GSM, etc.)
- Ofrece diferentes calidades en la entrega de mensajes
  - 0: Entrega de mensajes como mucho una vez
  - 1: Entrega asegurada (puede que por duplicado)
  - 2: Entrega asegurada una única vez

MQTT – Formato del mensaje

- Cabecera fija de 2 bytes

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

  - MSB seguido de LSB

- Cabecera de longitud variable
  - Sólo para algunos mensajes
- Campo de datos
  - Sólo para CONNECT, SUBSCRIBE y SUBACK

MQTT – Formato del mensaje

- Tipo de mensaje

Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request
PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

MQTT – Formato del mensaje

- Flags
  - DUP
    - Fijada cuando se reenvia un mensaje de tipo PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE.
    - El valor del flag QoS es mayor que 0
    - Requiere de un reconocimiento
    - La cabecera variable contiene un ID de mensaje
  - QoS – Para los mensajes de tipo PUBLISH

QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	<=1
1	0	1	At least once	Acknowledged delivery	>=1
2	1	0	Exactly once	Assured delivery	=1
3	1	1	Reserved		

  - RETAIN – Sólo para los mensajes de tipo PUBLISH
    - El cliente que lo publica
    - El servidor a los nuevos subscriptores

MQTT – Formato del mensaje

- Longitud del mensaje
  - Exceptuando la cabecera fija
  - 1 a 4 bytes
    - 7 bits de cada byte para la longitud
    - El bit 8 es un “continuation bit”

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

MQTT – Formato del mensaje

- Cabecera variable: Sucesión de cabeceras dependiendo de los mensajes
  - Protocol name (UTF-8): 1ª en los mensajes CONNECT
  - Protocol version (1 byte): 2ª en los mensajes CONNECT
  - Connect flags (1 byte): 3ª en los mensajes CONNECT
    - Bit 0 reservado
    - Clean session flag (bit 1) – Maneja la persistencia entre sesiones
    - Will flag (bit 2) – El cliente deja “testamento”
    - Will QoS (bits 3 y 4) – Nivel de QoS para el “testamento”
    - Will Retain (bit 5) – Indica si el “testamento” se debe retener
    - Password flag (bit 6) – Incluye password al conectar
    - User Name flag (bit 7) – Incluye nombre de usuario al conectar
  - Keep-alive timer (2 bytes): 4ª en los mensajes CONNECT

MQTT – Formato del mensaje

- Cabecera variable: Sucesión de cabeceras dependiendo de los mensajes
  - Connect return code (1 byte): 2ª en los mensajes CONNACK
  - Topic name: 1ª en los mensajes PUBLISH
    - Nombre del Topic en UTF-8
- Payload
  - CONNECT
    - ID de cliente (UTF-8): Obligatorio
    - Will Topic (UTF-8): Opcional
    - Will Message (UTF-8): Opcional
    - User Name (UTF-8): Opcional
    - Password (UTF-8): Opcional

MQTT – Formato del mensaje

- Payload
  - SUBSCRIBE
    - Topic Name (UTF-8): Obligatorio
    - QoS Level (1 byte): Obligatorio
    - Es posible subscribirse a varios Topic con el mismo mensaje
  - SUBACK
    - QoS Level Granted (1 byte): Obligatorio
    - Lista en el mismo orden que el SUBSCRIBE correspondiente

MQTT – Tipos de mensaje

- CONNECT
  - Cliente a Servidor
  - Crea la sesión de nivel de aplicación

Cabecera Fija	Cabecera Variable				Payload				
	Protocol Name (UTF-8)	Protocol Version (1 byte)	Connect Flags (1 byte)	Keep-alive Timer (2 bytes)	Client ID (UTF-8) (3 – 25 bytes)	Will Topic (UTF-8)	Will Message (UTF-8)	User Name (UTF-8)	Password (UTF-8)

- CONNACK
  - Servidor a Cliente
  - Respuesta al mensaje CONNECT

Cabecera Fija	Cabecera Variable	
	Reservado (1 byte)	Connect return code (1 byte)

MQTT – Tipos de mensaje

- PUBLISH
  - Cliente a Servidor y Servidor a Cliente
  - Información que se quiere distribuir
  - Asociado a un Topic (a.k.a. Subject o Channel)

Cabecera Fija	Cabecera Variable		Payload
	Topic Name (UTF-8)	Message ID (2 bytes)	Data (max. 256 MB)

MQTT – Tipos de mensaje

- PUBACK - Publish acknowledgment
  - Respuesta al mensaje PUBLISH con QoS = 1
  - Servidor a Cliente
    - Hacia el productor del mensaje
  - Cliente a Servidor
    - Desde un subscriptor

Cabecera Fija	Cabecera Variable
	Message ID (2 bytes)

MQTT – Tipos de mensaje

- PUBREC - Assured publish received
  - Respuesta al mensaje PUBLISH con QoS = 2
  - Servidor a Cliente
    - Hacia el productor del mensaje
  - Cliente a Servidor
    - Desde un subscriptor

Cabecera Fija	Cabecera Variable
	Message ID (2 bytes)

MQTT – Tipos de mensaje

- PUBREL - Assured Publish Release
  - Respuesta al mensaje PUBREC
  - Servidor a Cliente
    - Hacia el consumidor del mensaje PUBLISH con QoS = 2
  - Cliente a Servidor
    - Desde el productor del mensaje PUBLISH con QoS = 2

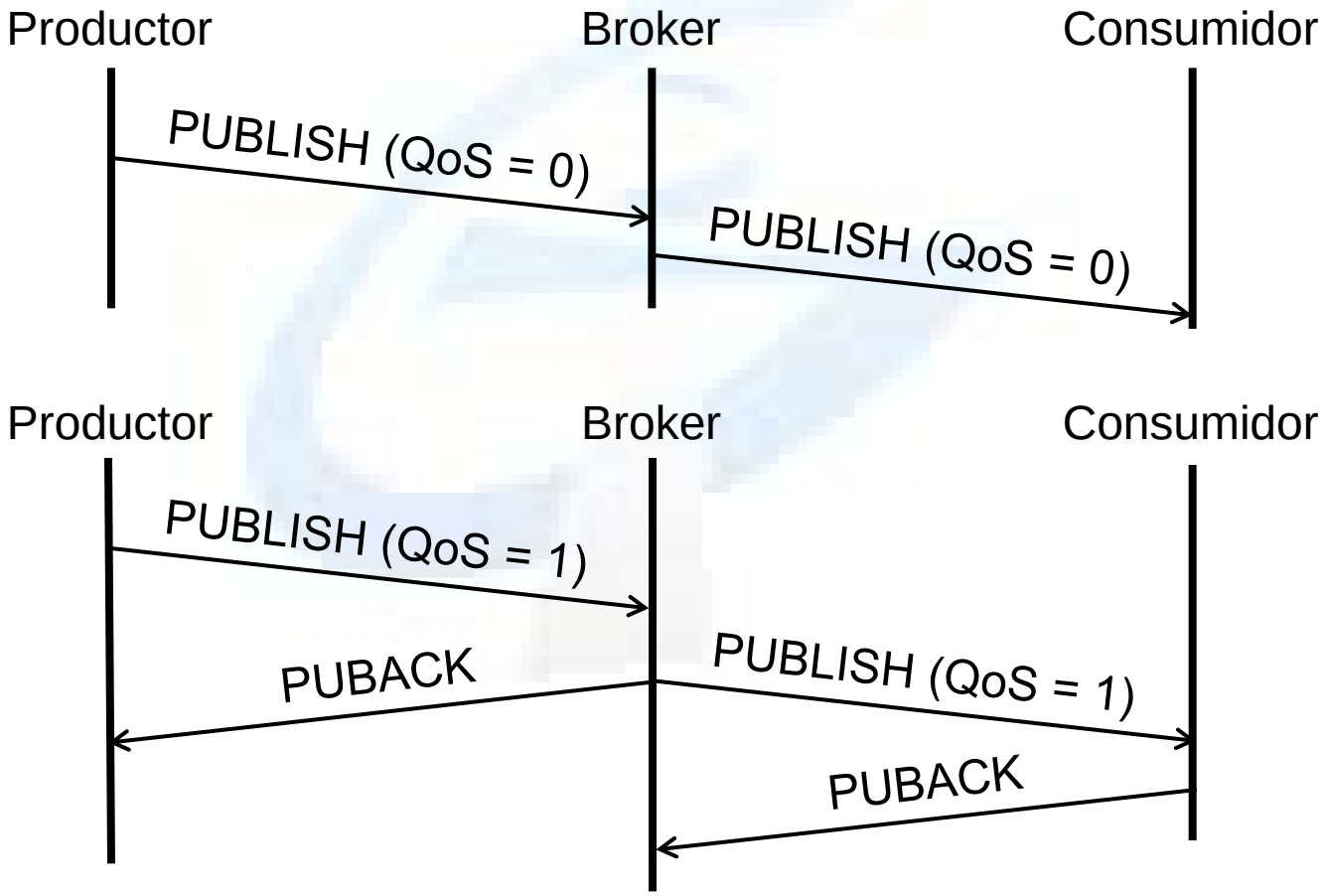
Cabecera Fija	Cabecera Variable
	Message ID (2 bytes)

MQTT – Tipos de mensaje

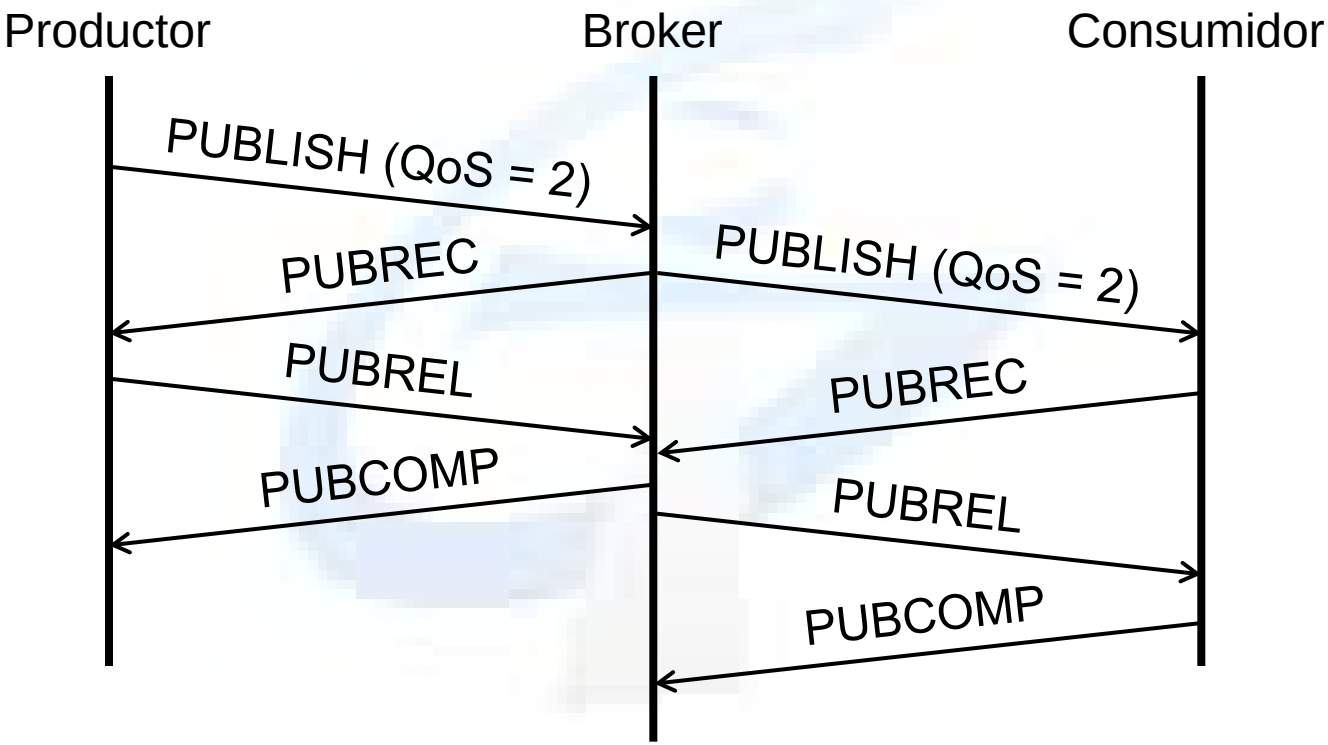
- PUBCOMP - Assured publish complete
  - Respuesta al mensaje PUBREL
  - Servidor a Cliente
    - Hacia el productor del mensaje
  - Cliente a Servidor
    - Desde un subscriptor

Cabecera Fija	Cabecera Variable
	Message ID (2 bytes)

MQTT – Tipos de mensaje



MQTT – Tipos de mensaje



MQTT – Tipos de mensaje

- SUBSCRIBE
  - Cliente a Servidor
  - Especifica el nivel de QoS

Cabecera Fija	Cabecera Variable	Payload						
	Message ID (2 bytes)	Topic Name1 (UTF-8)	QoS Level (1 byte)	Topic Name2 (UTF-8)	QoS Level (1 byte)	…	Topic NameN (UTF-8)	QoS Level (1 byte)

- SUBACK
  - Respuesta al mensaje SUBSCRIBE
  - Servidor a Cliente

Cabecera Fija	Cabecera Variable	Payload			
	Message ID (2 bytes)	Granted QoS (1 byte)	Granted QoS (1 byte)	…	Granted QoS (1 byte)

MQTT – Tipos de mensaje

- UNSUBSCRIBE
  - Cliente a Servidor
  - Especifica los Topic de los que se quiere des-registrar

Cabecera Fija	Cabecera Variable	Payload			
	Message ID (2 bytes)	Topic Name1 (UTF-8)	Topic Name2 (UTF-8)	...	Topic NameN (UTF-8)

- UNSUBACK
  - Respuesta al mensaje UNSUBSCRIBE
  - Servidor a Cliente

Cabecera Fija	Cabecera Variable
	Message ID (2 bytes)

MQTT – Tipos de mensaje

- PINGREQ
  - Cliente a Servidor
  - Mensaje de keep-alive
- PINGRESP
  - Respuesta al mensaje PINGREQ
  - Servidor a Cliente
- DISCONNECT
  - Cliente a Servidor
  - Antes de cerrar la conexión TCP

MQTT – Flujos

- Nivel de QoS = 0

Cliente	Mensaje y Dirección	Broker
QoS = 0	PUBLISH ---------->	**Acción:** Publicar el mensaje a los subscriptores

- Nivel de QoS = 1

Cliente	Mensaje y Dirección	Broker
QoS = 1 DUP = 0 Message ID = X **Acción:** Guardar el mensaje	PUBLISH ---------->	**Acciones:** • Guardar el mensaje • Publicar el mensaje a los subscriptores • Borrar el mensaje
Acción: Eliminar el mensaje	PUBACK <----------	

MQTT – Flujos

- Nivel de QoS = 2

Cliente	Mensaje y Dirección	Broker
QoS = 1 DUP = 0 Message ID = X **Acción:** Guardar el mensaje	PUBLISH ---------->	**Acción:** Guardar el mensaje
	PUBREC <----------	Message ID = X
Message ID = X	PUBREL ---------->	**Acciones:** • Publicar el mensaje a los subscriptores • Borrar el mensaje
Acción: Eliminar el mensaje	PUBCOMP <----------	Message ID = X

MQTT – Implementaciones

- La especificación es abierta
  - http://www.ibm.com/developerworks/webserices/library/ws-mqtt/index.html
- Multiples implementaciones disponibles
  - Desde librerías profesionales hasta proyectos individuales
- Librerías ligeras (30 Kb en C; 64 Kb en Java)
  - http://mqtt.org
- API fácil de usar a través de verbos bien definidos

MQTT – Beneficios frente a HTTP

- Entrega de mensajes tipo “push”
  - HTTP soporta “push” del cliente hacia el servidor pero del servidor al cliente es tipo “poll”
- Uso eficiente de la red
  - Volumen de tráfico mucho menor
- Soporta las desconexiones
- Paradigma uno a muchos

		3G		WiFi	
		HTTPS	MQTT	HTTPS	MQTT
rx	msg / hora	1708	160278	3628	263314
	% bateria / msg	0.01709	0.0001	0.00095	0.00002
	msgs	240 / 1024	1024 / 1024	524 / 1024	1024 / 1024
tx	msg / hora	1926	21685	5229	23184
	% bateria / msg	0.00975	0.00082	0.00104	0.00016

MQTT – Referencias

- Especificación abierta
  - http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html
- Comunidad
  - http://mqtt.org
- Estandar
  - https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt