

Διαχείριση Edge και Cloud δικτύων βασισμένων στο λογισμικό (CSIS109)

Δρ. Ειρήνη Λιώτου

eliotou@hua.gr

6/5/2025

Internet of Things

- The Internet of Things (IoT) encapsulates a vision of a world in which billions of objects with embedded intelligence, communication means, and sensing and actuation capabilities will connect over IP networks.
- These “things” or “objects”:
 - can be worn by users or deployed in the environment
 - gather data from the environment and act upon it
 - usually highly constrained, with limited memory and available energy stores
 - subject to stringent low-cost requirements

Edge Computing paradigm

- Introducing computing resources at the edge of access networks may bring several benefits that are key for IoT scenarios: low latency, real-time capabilities and context-awareness.
- Edge nodes (servers or micro data-centers on the edge) may act as an interface to data streams coming from connected devices, objects, and applications.
- The stored Big Data can then be processed with new mechanisms, such as machine and deep learning, transforming raw data generated by connected objects into useful information.
- The useful information will then be disseminated to relevant devices and interested users or stored for further processing and access.

Wireless Ad-hoc and Sensor Networks: The Ancestors

- WSNs promised wide support of interactions between people and their surroundings:
 - “Wireless”: mobility support and ease of system deployment;
 - “Sensor”: capability to perceive and interact with the world;
 - “Networks”: plurality of communicating devices.
- Sources placed around the areas to be monitored and the sinks located in easily accessible places.
- Sources send information to sinks in accordance with different scheduling policies: time-driven, event-driven, query-driven.
- Synonymous to “ad-hoc networks” = general, infrastructure-less, cooperation-based, opportunistic networks, customized for specific scenarios.

Key challenges in WSNs/ad-hoc networks

- self-configuration and self-organization in infrastructure-less systems;
- support for cooperative operations in systems with heterogenous members;
- multi-hop peer-to-peer communication among network nodes, with effective routing protocols;
- network self-healing behavior providing a sufficient degree of robustness and reliability;
- seamless mobility management and support of dynamic network topologies.

IoT-enabled Applications

- Home and Building Automation (smart home): home security and energy efficiency, light and room control
- Smart Cities: healthcare, media, energy and the environment, safety, and public services
- Smart Grid: smart meters, smart appliances, renewable energy resources, and energy-efficient resources
- Industrial Internet of Things (IIoT): manufacturing, logistics, oil and gas, transportation, energy/utilities, mining and metals, aviation and others
- Capabilities:
 - sensor-driven computing: converting sensed data into insights that operators and systems can act on;
 - industrial analytics: turning data from sensors and other sources into actionable insights;
 - intelligent machine applications: integrating sensing devices and intelligent components into machines.

IoT-enabled Applications

- Smart Farming (precision farming or smart agriculture): applications for monitoring the weather, automation for more precise application of fertilizers and pesticides, and the adoption of planning strategies for maintenance.
- Drones and sensor networks (to collect data) and cloud platforms (to manage the collected data).
- Fleet management, livestock monitoring, fish farming, forest care, indoor city farming.
- Aim at supporting farmers in their decision processes through decision-support systems => less waste and increase in efficiency.

Interoperability challenges

- Current IoT applications have prior knowledge of the things they will work with: there are established communication protocols, data formats, and application-specific interaction patterns
- Smart objects manufactured by the same vendor typically need to be accessed through legacy software, resulting in a plethora of applications that end-users must install and use.
- Mobility challenge if users want to fully and seamlessly interact with things while moving / entering other environments.
- Smart objects should be able to adapt dynamically to particular conditions, such as a change in their battery level or hosted resources, or the presence of specific users => require upgrades or new software to be installed

IoT Architecture

- In order to avoid dependence on specific legacy software all actors need to speak the same language: IP =>
 - Define open standards for communication (e.g., 6LoWPAN/CoAP);
 - Map the traditional IP-based Internet stack to the IoT.
- The IoT will be a network of heterogeneous interconnected devices. This will be the infrastructure for the so-called “Web of Things” (WoT).
- CoAP has been designated as the standard protocol for the WoT, similar to the position of HTTP for the web.

Physical/Link Layer

- IEEE 802.15.4 standard and Zigbee
 - 868.0–868.6 MHz: used in Europe
 - low bit-rate connectivity
 - typically between 10 and 100 m
- Low-power Wi-Fi (802.11ah / Wi-Fi HaLow)
- Bluetooth and its newest energy-efficient version Bluetooth Low Energy (BLE)
 - 50Mbps, 240m
- Protocols in the area of power line communications (PLC)

Network Layer

- The depletion of IPv4 addresses makes it impossible to assign public IPv4 addresses to objects.
- NAT would be needed, which would involve complex configuration management to ensure reachability of smart objects.
- => The only feasible solution to create a global, sustainable, and scalable IoT is to adopt IPv6 at the network layer (3.4×10^{38} unique IP addresses).
- Low-power wireless personal area networks (WPANs) have special characteristics => an adaptation layer that fits IPv6 packets to the IEEE 802.15.4 specifications is needed.
- The goal of **6LoWPAN** is to transmit a small IPv6 datagram over a single IEEE 802.15.4 hop.

Transport/Application Layer

- Transport: IoT scenarios call for energy-efficient, lightweight, and non CPU-intensive approaches to communication => UDP
- Application: Dedicated web transfer protocol for low-power and lossy networks (LLNs), called the Constrained Application Protocol (CoAP), has been defined.

The Verticals: Cloud-based Solutions

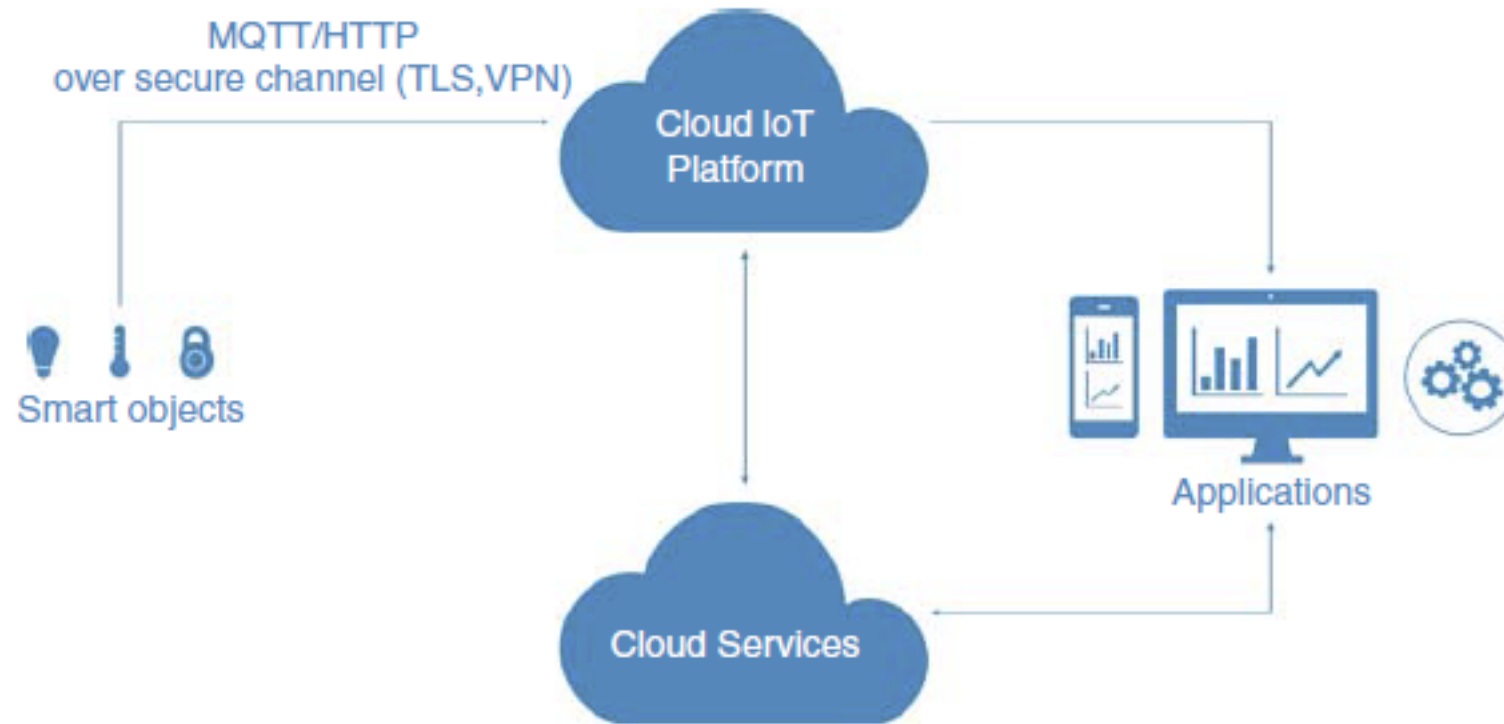


Figure 3.1 Cloud IoT platform architecture.

- Amazon's AWS IoT and Microsoft's Azure IoT suite are probably the most popular cloud IoT platforms

Requirements for a “long-term” design

- Scalability
 - Availability
 - Interoperability
 - Security
 - Evolution of systems
-
- ... So this approach cannot be the reference architecture for a scalable and evolutionary IoT.

REST Architectures: The Web of Things

- It should be built in a similar way to the Internet (web-based approach)!
 - Referencing of resources through uniform resource identifiers (URIs)
 - Introduction of the Hypertext Transfer Protocol (HTTP)
- REpresentational State Transfer (REST) is the architectural style behind the web
- REST defines a set of rules and principles that all the elements of the architecture must conform to in order to build web applications that scale well, in terms of scalability and robustness
- Loose coupling means that the endpoints should contain as little information about each other as needed to work. All necessary missing information should be collected while interacting.

Resource-oriented architectures

- REST is based on the concept of a resource. A resource can be defined as any relevant entity in an application's domain that is exposed on the network.
- A webpage, a video, and an order on an e-commerce website can all be considered web resources.
- A resource is anything with which a user interacts while progressing toward some goal.

Service-oriented architecture

- An alternative to a resource-oriented architecture (ROA) is a service-oriented architecture (SOA).
- SOA refers to an architecture where two endpoints communicate through a pre-defined set of messaging contracts. A client starts interacting with a server by retrieving the list of available services and how these can be mapped to HTTP messages, in a Web Service Definition Language (WSDL) document.
- However, this is a weakness: if a server changes its services, a client needs to get access to the new WSDL or its functionalities are invalidated.

REST architectures

- A REST architecture builds on:
 - clients (or user agents, such as browsers), which are the application interface and initiate the interactions
 - servers (origin servers) host resources and serve client requests.
- A REST architecture is characterized by uniform interfaces: all connectors within the system must conform to this interface's constraints.
- Collectively, REST defines the following principles:
 - identification of resources
 - manipulation of resources through representations
 - self-descriptive messages
 - hypermedia as the engine of application states.
- An application that follows the above principles is termed RESTful.

Representation of resources

- Resources are never exchanged directly by endpoints. Instead, representations of resources are exchanged between endpoints.
- A representation is a view of the state of the resource at a given time. This view can be encoded in one or more transferable formats, such as XHTML, Atom, XML, JSON, plain text, comma-separated values, MP3, or JPEG.

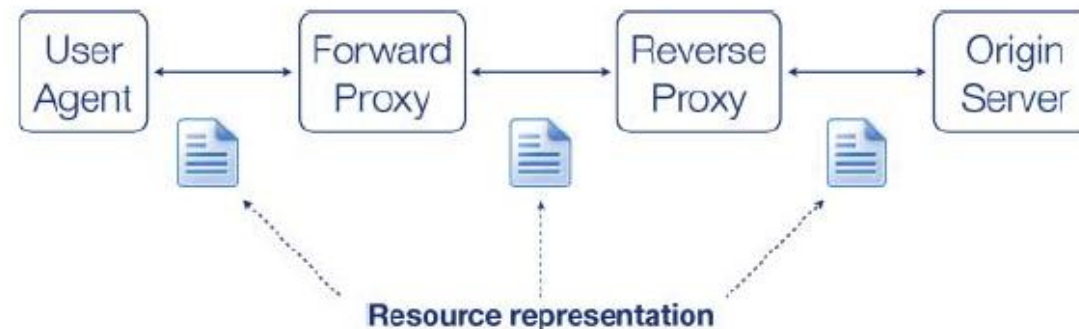


Figure 3.2 Representations of resources are exchanged between endpoints.

Resource identifiers

- Uniform resource identifiers (URIs) identify a resource univocally.
- A URI can be used to address a resource, so that it can be located, retrieved, and manipulated.
- There is a 1:N relationship between a resource and URIs: a resource can be mapped to multiple URIs, but a URI points exactly to one resource.
- URIs can be of two kinds:
 - a uniform resource name (URN) specifies the name of a resource (e.g., urn:ietf:rfc:2616);
 - a uniform resource locator (URL) specifies how to locate the resource, (e.g., <http://example.com/books/123>).
- All URIs take the following form:
 - scheme:scheme-specific-part.

http://	example.com	:8080	/people	?id=1	#address
scheme	host	port	path	query	fragment

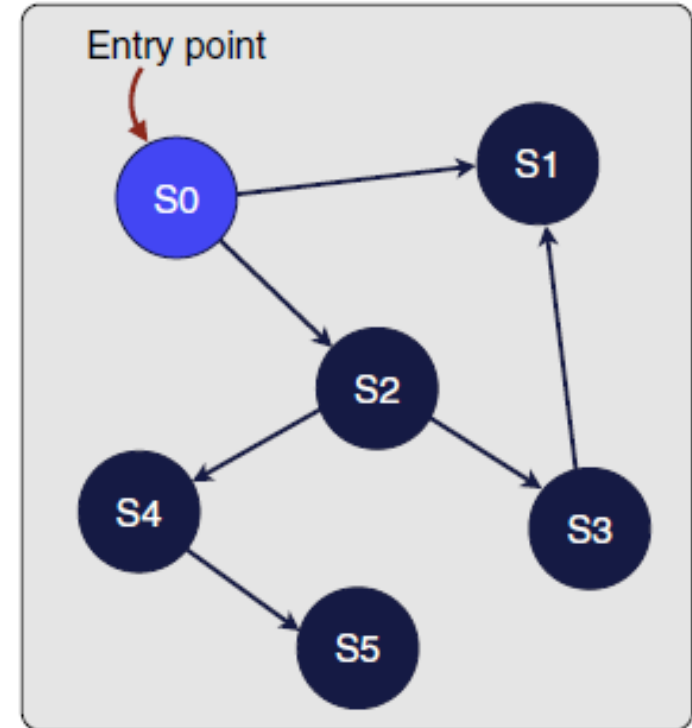
Figure 3.3 Generic URL structure.

Statelessness

- Statelessness implies that no state information must be kept on the client and server sides, thus avoiding the need to use cookies or to introduce the concept of sessions, which demand a stricter coupling between the endpoints.
- In order to preserve statelessness, each message must be self-descriptive.
- This means that all requests must contain all the information to understand the request so that servers can process them without context (about the state of the client).

Applications as Finite-state Machines

- RESTful applications make forward progress by transitioning from one state to another, just like a finite-state machine (FSM).
- A state is reached when the server transfers a representation of the resource because of a client request. The next possible transitions are discovered when the application reaches a new state (gradual reveal).
- Resource representations that embed links are called hypermedia. These links represent the possible transitions to the next states.
- In essence, the state of a resource identified by a URI is contained in the data section of the resource representation and the transition to the next states are contained in the links.



Hypermedia

- The final principle of REST is “hypermedia as the engine of application state”, or HATEOAS.
- RESTful applications progress according to the following steps:
 1. The client starts from an entry URI or a bookmark.
 2. The response to a GET request includes a hypermedia representation.
 3. The representation contains links that define the possible transitions to the next states of the FSM.
 4. The client selects a link to follow and issues the next request; that is, it triggers a transition to the next state.
 5. The client can also go back.

Richardson Maturity Model

- The Richardson maturity model is a classification system for web-based applications.
- It can be used to answer the question “How RESTful is a web application?”
- The higher the level, the more RESTful an application is: the higher the level, the less coupling exists between clients and servers.

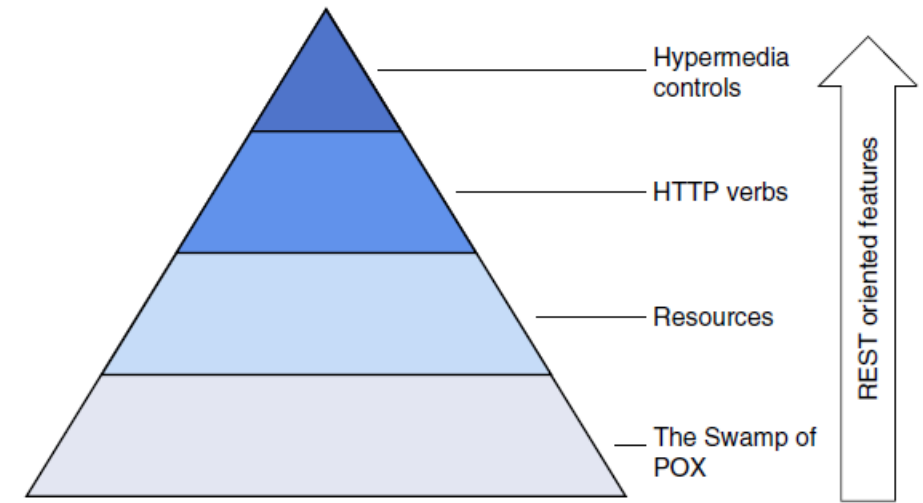


Figure 3.5 The Richardson maturity model.

Level 0: the Swamp of POX (plain old XML)

- All the semantics of an interaction are strictly tied to the syntax that clients and servers use.
- The client must have a very deep a-priori knowledge of:
 - the web service
 - the actions that can be triggered
 - the meaning of XML document tags and attributes.
- If the web service changes something, the client just breaks.

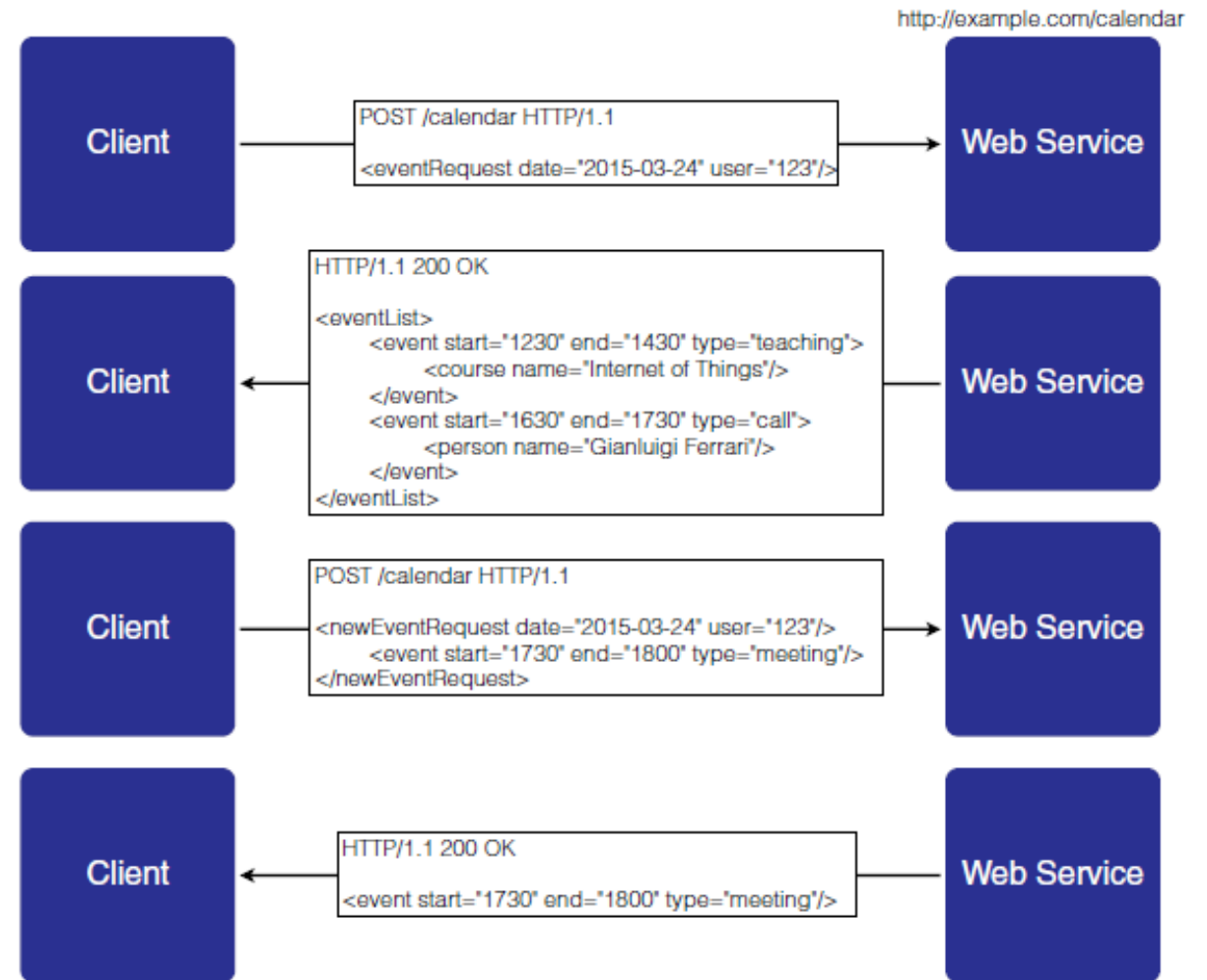


Figure 3.6 XML-based RPC.

Level 1: Resources

- It models interactions by targeting resources instead of services.
- Applications that use the concept of resources rather than services are classified as Level 1.
- Action names and parameters are typically mapped directly to a URI, rather than embedded in the semantics of XML/SOAP payloads.
- The action is triggered by sending an HTTP GET or POST request to the targeted URI, for example:
 - GET `http://example.com/people/123?action=delete`

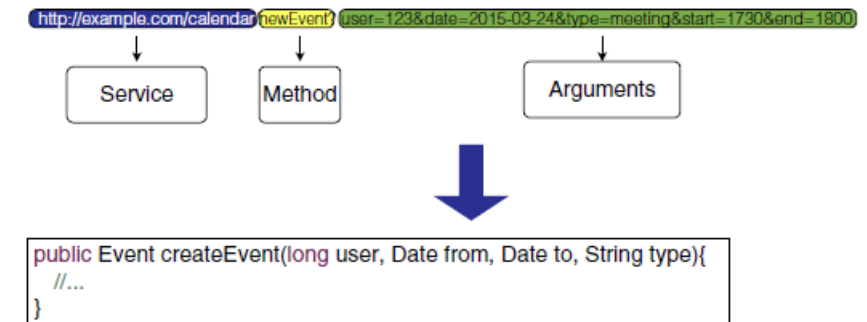





Figure 3.7 URI-to-action mapping.





Level 2: HTTP Verbs

- It moves the semantics from the URI to HTTP verbs when manipulating resources.
- Resources are still addressable using URIs, and each resource can be manipulated using HTTP methods. Each method has a particular meaning and maps to a specific **CRUD (create-read-update-delete)** operation.
- According to RFC 2616, HTTP methods can be safe and/or idempotent.
 - Safe means that they have no effect on the resource (the resource remains the same).
 - Idempotent means that the same request can be executed multiple times with the same effect as executing it once.

“Safe” concept

CRUD Operation	HTTP Method	Safe?	Description
Create	POST	 No	Creates a new resource.
Read	GET	 Yes	Retrieves a resource; read-only.
Update	PUT	 No	Replaces an existing resource.
Delete	DELETE	 No	Removes a resource.

“Idempotent” concept

Method	Idempotent?	Why
GET	 Yes	Retrieves data, doesn't change anything.
PUT	 Yes	Replaces a resource — doing it again just replaces it with the same data.
DELETE	 Yes	Deleting something that's already gone doesn't change the state further.
POST	 No	Creates a new resource — sending it twice usually creates duplicates.

1. Creating a resource

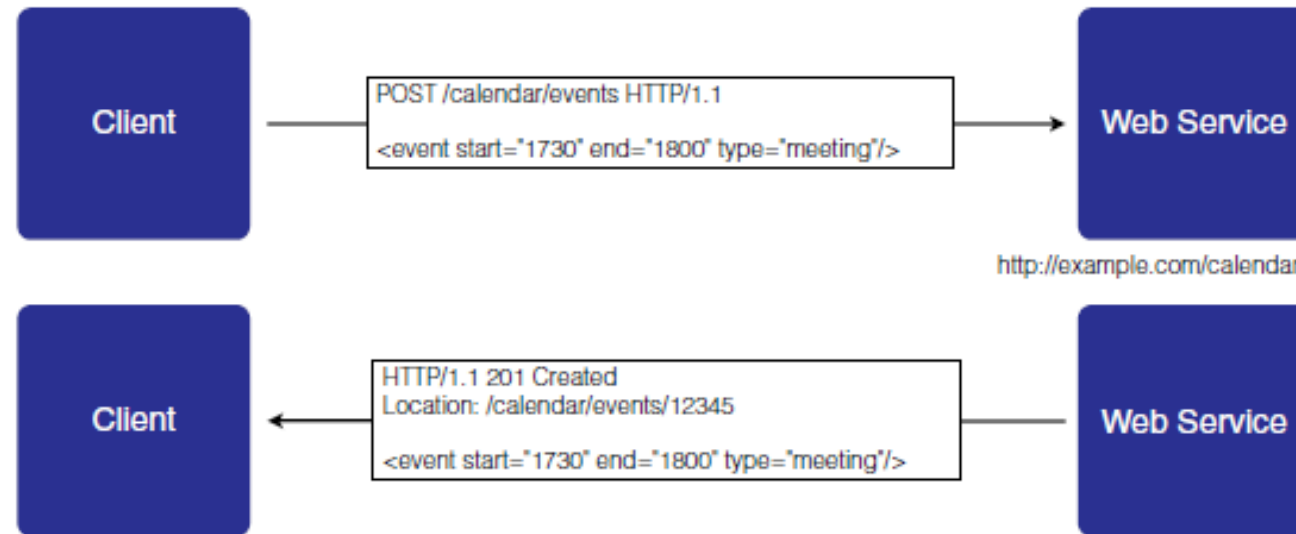


Figure 3.8 Resource creation.

2. Retrieving a resource

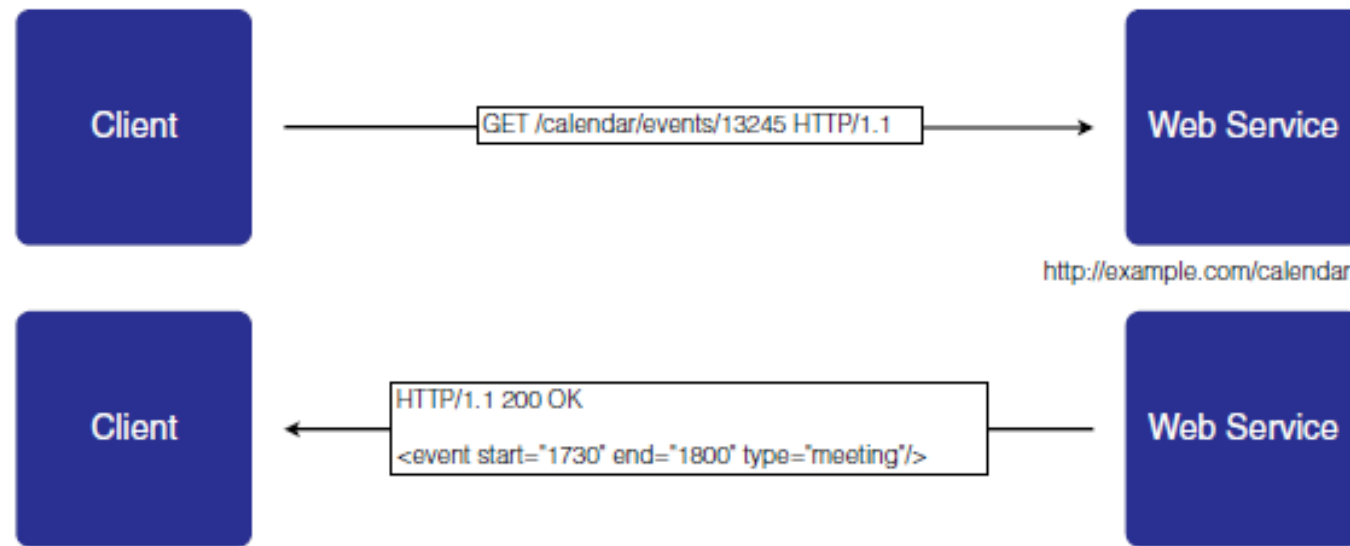


Figure 3.9 Resource retrieval.

3. Updating a resource

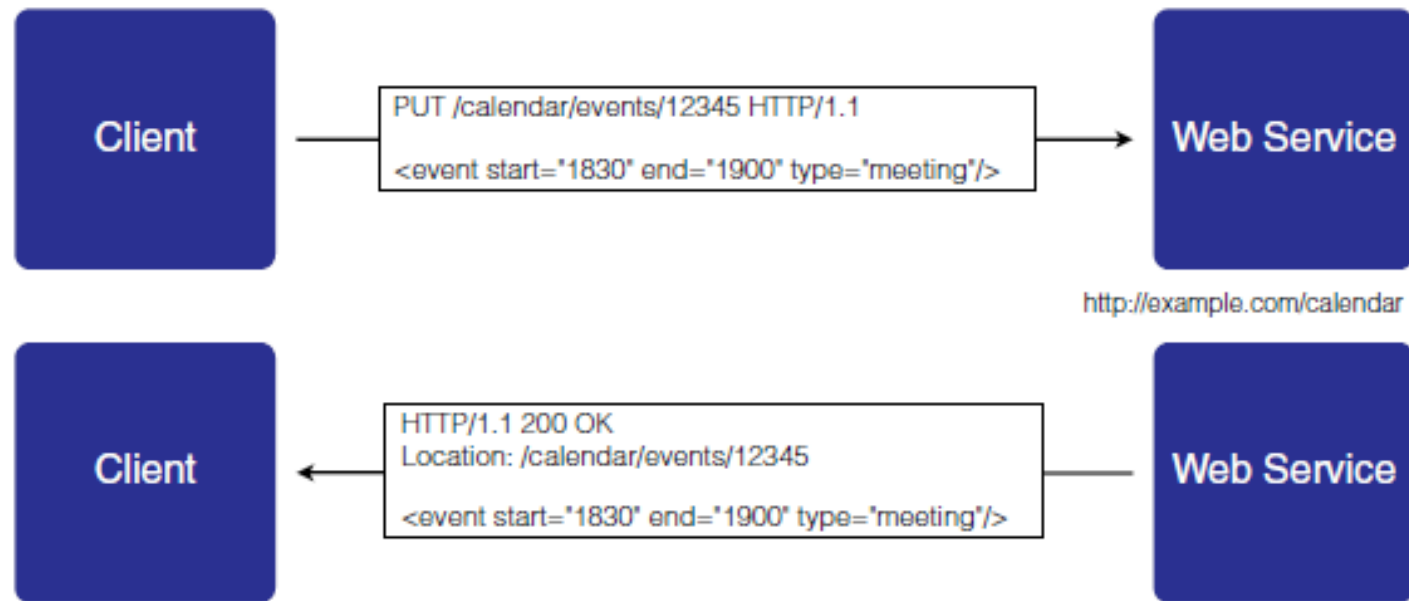


Figure 3.10 Resource update.

4. Deleting a resource

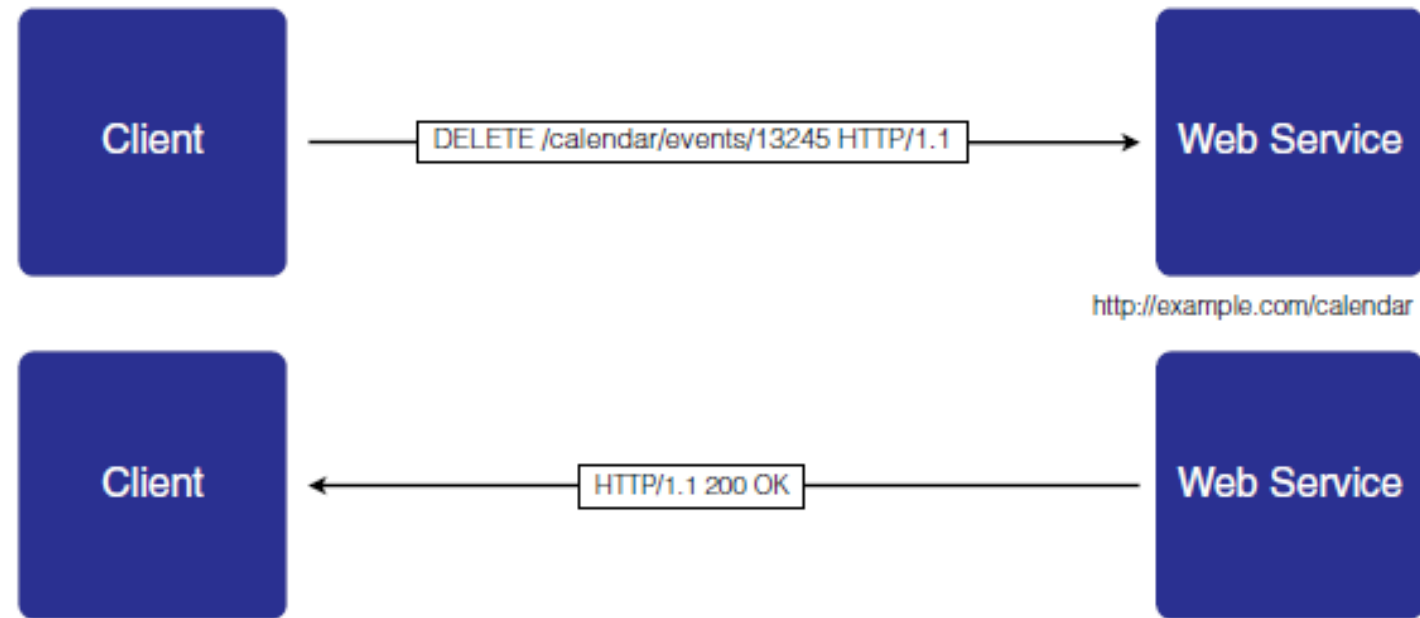


Figure 3.11 Resource deletion.

HTTP Response Code Semantics

Table 3.2 Semantics of 2xx HTTP status codes when manipulating resources.

Method	Status code	Reason
GET	200	OK
POST	201	Created
PUT	200	OK
	204	No content
	200	OK
	202	Accepted
DELETE	202	Accepted
	204	No content

Table 3.3 Semantics HTTP error status codes when manipulating resources.

Status code	Reason	Code
400	Bad request	The client isseud a malformed request
401	Unauthorized	The client must authenticate before performing the request
403	Forbidden	The client does not have the privileges to access the resource
404	Not found	No resource with the given URI was found
405	Method not allowed	The resource cannot be manipulated using the HTTP method
500	Internal server error	The server failed to process the request

Describing Level 2 Applications

- HTTP is no longer used just as a transport for requests, but instead is also used to describe what manipulation on the resource is being requested.
- Level 2 applications use:
 - HTTP verbs and status codes to coordinate interactions and manipulate resources;
 - HTTP headers to convey information (e.g., the Location header to indicate the URI of a created resource).
- However, there is still some coupling between client and server applications: the client must know the URI of a resource and which methods can be invoked.

WADL

- The solution to this problem is to use a Web Application Description Language (WADL) document.
- A WADL document is a static description used to advertise the endpoints, the methods, and the representation formats of the resources hosted by a web application. WADL documents describe:
 - sets of resources;
 - relationships between resources;
 - methods that can be applied to each resource, together with expected input/output and formats;
 - resource representation formats (MIME types and data schemas).

Level 3: Hypermedia

- HATEOS principle: ultimate guarantee that client and server applications are fully decoupled
- Hypermedia embed links to drive application states
- When clients reach a state of the application, the representation of the resource has a double goal:
 - it describes the current state;
 - it includes link information to drive the client perform the next intended transitions, according to what the server expects.
- The state of a resource is the aggregation of:
 - data: values of information items belonging to that resource;
 - links: representing transitions to possible future states of the current resource.

Level 3: Hypermedia

- A client capable of understanding the meaning of the hypermedia is fully autonomous in the execution of all the operations, regardless of any change on the server.
- A server can change the URI scheme independently without breaking clients and can introduce new functionalities (states) just by adding more links in the hypermedia.
- HATEOAS fully enables true independent evolution of systems.
- An IoT standard for hypermedia to be used in constrained environments is the CoRE Link Format

Summary (ChatGPT)

Summary Table (All Levels)

Level	URI (Resources)	HTTP Verbs	Hypermedia (Links)
0	✗	✗	✗
1	✓	✗	✗
2	✓	✓	✗
3	✓	✓	✓

The Web of Things

- Modeling the IoT using web-oriented, RESTful principles can be a way to start to develop a global infrastructure of interconnected objects and to foster the development of scalable and robust IoT applications.
- The basic idea is to consider smart objects as tiny servers that implement Level 3 IoT applications using hypermedia and the CoRE Link Format.
- Full interoperability between the web and the WoT.

Messaging Queues and Publish/Subscribe Communications

- Messaging systems implement an asynchronous communication model and loosely couple the senders with the consumers of messages, thus allowing for more flexibility and scalability.
- Messaging systems implement one of two asynchronous messaging approaches: message queues or pub/sub.
- Message queues: The sender sends a message to a queue on a server, where it is stored/persistent: the message is not erased immediately but is kept in memory until a consumer receives it. Only once delivered is the message deleted from the queue.

Publish/Subscribe

- Two kinds of entities exist: publishers and subscribers. Publishers send messages to a “topic” on the server.
- Subscribers can subscribe to a topic to receive a copy of all messages that have been published on that topic. This means that a message can be consumed by multiple consumers.

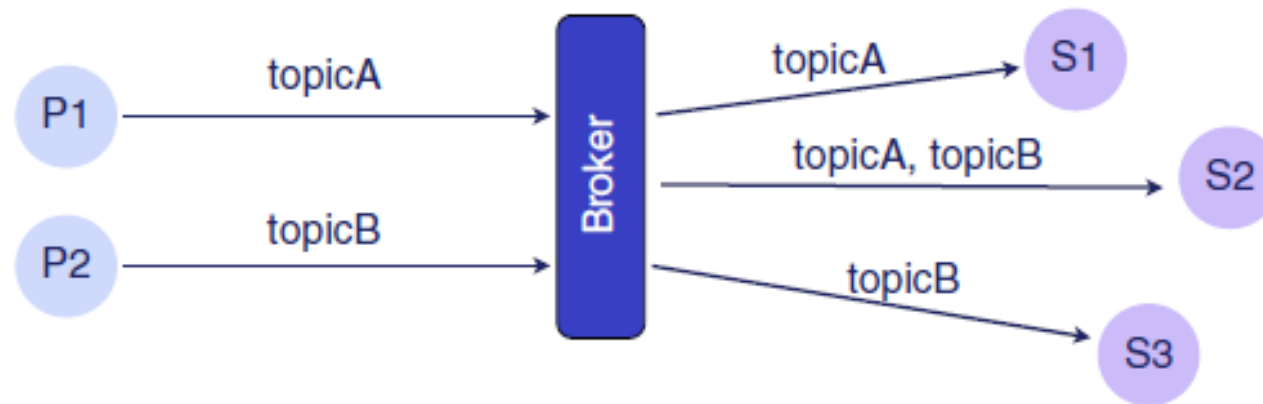


Figure 3.12 Publish/subscribe communication model.

Publish/Subscribe

- The separation between publishers and subscribers is possible thanks to intermediary nodes, called brokers.
- Brokers can be implemented as message queues. Typically, the broker is involved for the following functions:
 - publishing: publishers send messages to the broker;
 - subscriptions: subscribers register to receive messages.
- Upon receiving a message from a publisher, the broker is responsible for dispatching messages to the subscribers, according to their subscriptions.

Publish/Subscribe pros & cons

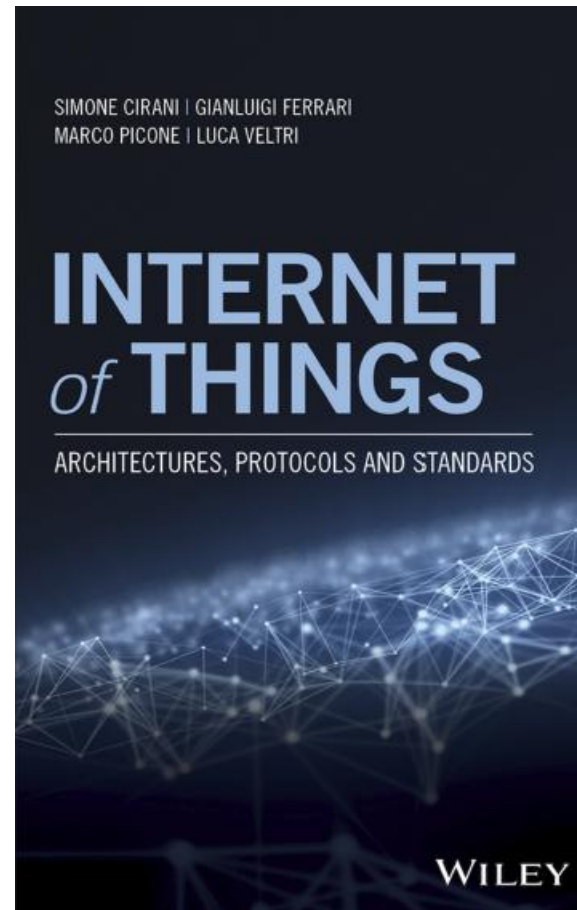
- + Loose coupling: Publishers need not know which subscribers receive messages or even if they exist.
- + Scalability: Since brokers only need to route messages, they can be replicated easily to support higher volumes of data being transferred.
- + Lightweight implementation: Most of the load is carried by the broker.
- - No content-type negotiation can be performed
- - No support for end-to-end security
- - the broker infrastructure must scale in order to avoid issues related to load peaks

Message Queue Telemetry Transport (MQTT)

- A lightweight, open-source, TCP-based pub/sub protocol.
- Particularly suited to constrained environments where message protocol overhead and message size should be minimal.
- Messages are published to a shared topic space inside the broker. Topics are used as filters on the message stream from all publishers to the broker.
- Messages are delivered to all clients that have subscribed with a matching topic filter.
- The broker applies the subscription filters to the message stream it receives in order to efficiently determine to which clients a message should be dispatched.

References

Chapters 1, 2, 3



Thank you!