Cloud Applications

Cloud application design methodologies

Reference:

- [cam-san] Chapter 12
- Additional material: ***Distributed and cloud computing - Chapter 5.1 by Kai Hwang, Geoffrey C. Fox, Jack J. Dongarra***

Cloud application design

- Cloud applications are distributed systems, made of different components interacting each other to eventually offer an application to users
- The implementation and integration of different software components for the cloud environment is a complex task that can result to be time consuming and expensive
- In order to avoid long time to market, design and development of software objects should follow a certain rigorous methodology
- Such methodology should embed main requirements, mandatory in a dynamic cloud environment

Cloud application design requirements

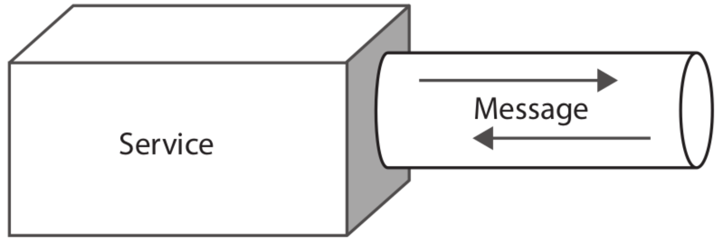
- A software component, part of a cloud application, should be designed with the following requirements
  - **Interoperability**: in the cloud a software component is meant to interact with other components. In order to ensure interoperability and integration, the software should expose a standard interface that can be easily invoked by other software/components of the application
  - **Scalability**: one of the main feature of the cloud infrastructure is horizontal scalability, the application should be designed in order to be ready to be integrated with a growing number of components, dynamically and effortlessly
  - **Composability**: a cloud environment is a dynamic environment that can change rapidly, e.g. due to failure of components, creation/destruction of VMs, etc. A software should be designed in order to allow its usage with different components without (or with minimal) changes

Service-Oriented Architecture

- In recent years, a specific methodology has emerged over the others as the most suited style for software design for cloud applications: the ***Service-Oriented Architecture*** (SOA)
- SOA is not a tool or a framework, it is instead a modular approach for designing software objects
- Software components in SOA are referred as services, which is a self-contained component that implements specific functionalities
- In SOA *applications are built as a collection of services (or microservices) from existing software components*
- *Applications are highly modular rather than a single massive program*
- Software components can be developed using *different programming languages* and run on *heterogeneous hosts* in a distributed manner

SOA - Architecture

- Services within the application communicate via **<u>messages passing</u>**
- Each message has a specific schema that defines the format and the exchange of information
- Messages are sent from one service to another, each message is handled internally by the service implementation, in a hidden manner
- Other approaches, like for instance the Remote Method Invocation (RMI), limit flexibility and interoperability as they are based on a technological dependency between one software component and another, e.g. a software component that implement the remote method invocation
- The message schema of SOA eliminates this dependency removing any technological dependency. Applications can be implemented in different languages, as they only need to implement the message schema and behave accordingly
- The message schema does not only define the service (what it does and what information provides) but it also describes the message exchange patterns to interact with the service to invoke a certain functionality or to get certain data

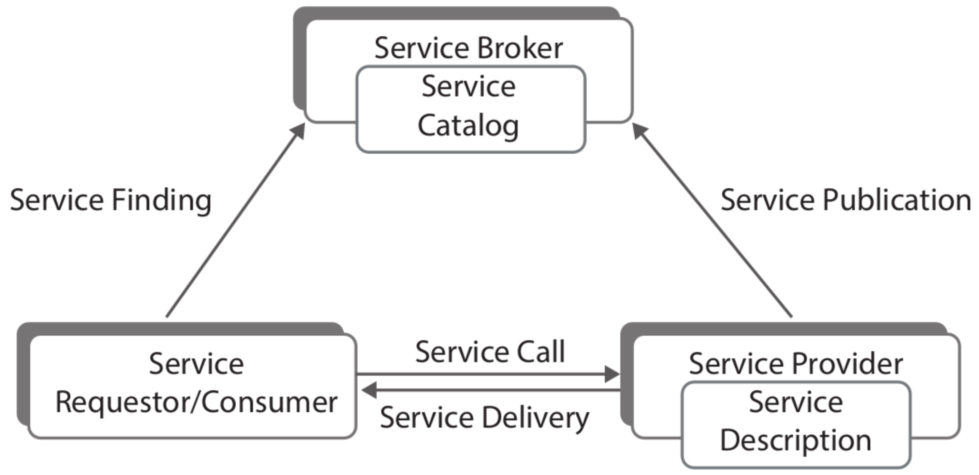


SOA - Roles

- In a SOA interaction we have two roles: a **consumer** service and a **provider**
- A consumer service sends a service request to a provider service via message passing
- The provider return a response to the consumer containing the expected results
- A service can always be a consumer, a provider, or it can be both, thus implementing one role or the other depending on the specific situation

SOA – Broker

- In general service providers and consumers do not interact directly with each other initially, as the consumer should know the address and the descriptor of the provider
- An additional software module is usually employed in the middle, a **<u>broker</u>**
- *The broker is responsible for creating a <u>service catalog (or registry)</u> in which all the available services are enlisted*
- *A client service can discover a provider service only if the interface is published to the broker*
- *After the consumer queried the broker for a specific service and obtained its identity, it can interact directly with the provider*
- *The address and the descriptor of the broker are well known*
- *As the interface and the address of a provider are known, the consumer can contact directly the provider*



SOA - Advantages

- The implementation of services remains hidden, thus making possible to fix the implementation without affecting other services as long as the service declaration/interface remains intact
- This separation enables a **loose coupling** feature that allows to plug and unplug specific services from an application effortlessly
- This feature makes the system flexible to changes and expandable, i.e. new features can be added effortlessly in short amount of time
- SOA also ensures **composability**, as a system can be assembled using a combination of existing services
- The message-based communication paradigm, ensure that communication among services is performed in a **stateless** manner: a module threats each transaction separately, in an independent manner of any other transaction

SOA - Scalability

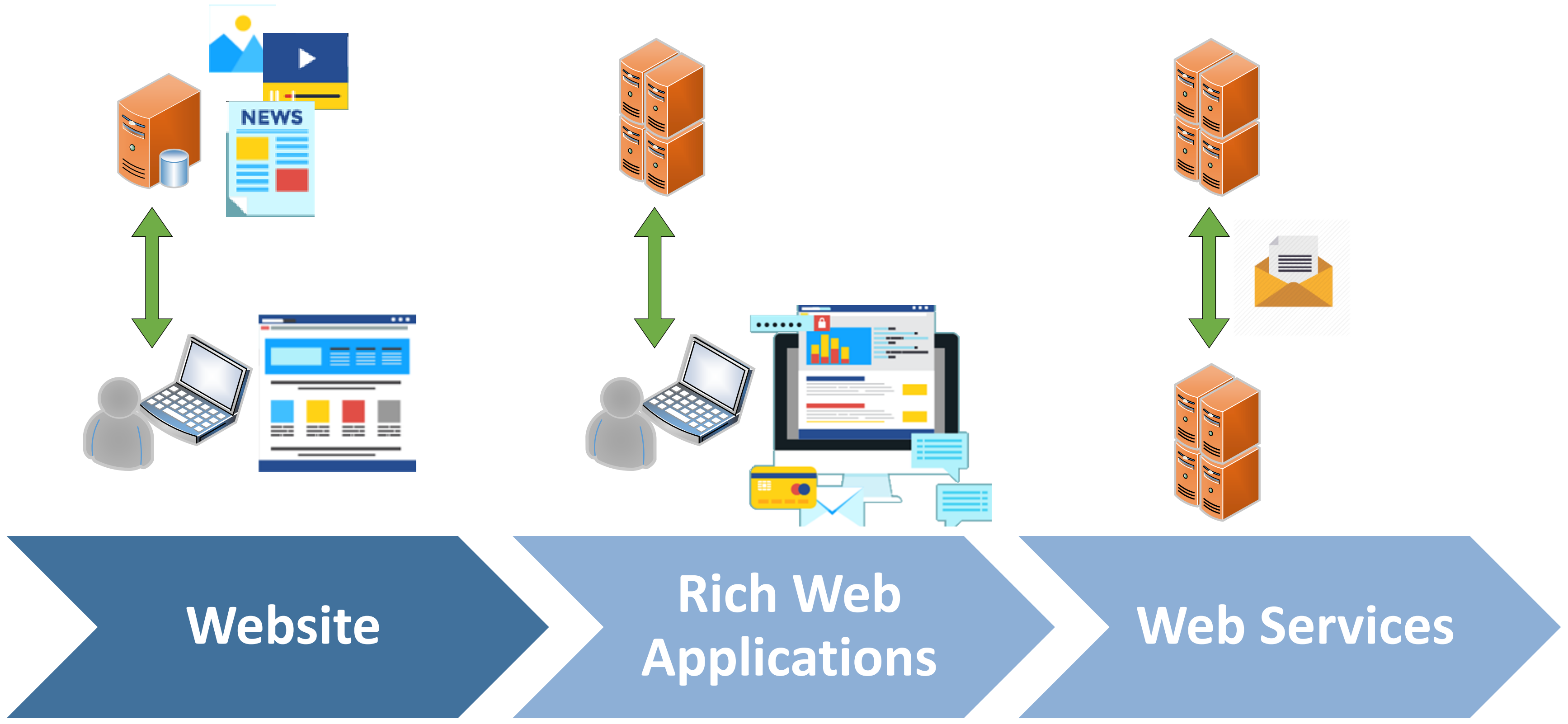
- SOA services that are *stateless and loosely coupled* are perfect to adapt to the scalability of the cloud infrastructure
- The broker in particular can help in managing software that are created or destroyed on demand and to pair them with consumers
- As a new instance of a service is created (e.g. because the number of requests increased and novel VMs are created with a specific service) they are registered to the broker and made available to consumer
- Vice-versa, as instances are destroyed (e.g. because the load decreased and some VMs are deallocated), they are unregistered and removed in a transparent manner

Web Services

- The Web has become a medium for connecting remote clients with applications
- The term Web Service (WS) is often referred to a self-contained, self-describing modular application designed to be used and accessible by other software applications across the web
- ***<u>Web Services are one of the most common instances of SOA implementations</u>***
- The W3C working group (the group behind the standardization of the WWW) defines a *web service a software system designed to provide interoperable machine-to-machine interaction over a network*

World Wide Web

- Web already provided mechanism for:
  - Identify resources
  - Transfer data
  - A markup language to encode information
- WS are the natural evolution and adoption of the SOA paradigm over the Web



- A (huge) collection of multimedia data

- A GUI to access network applications

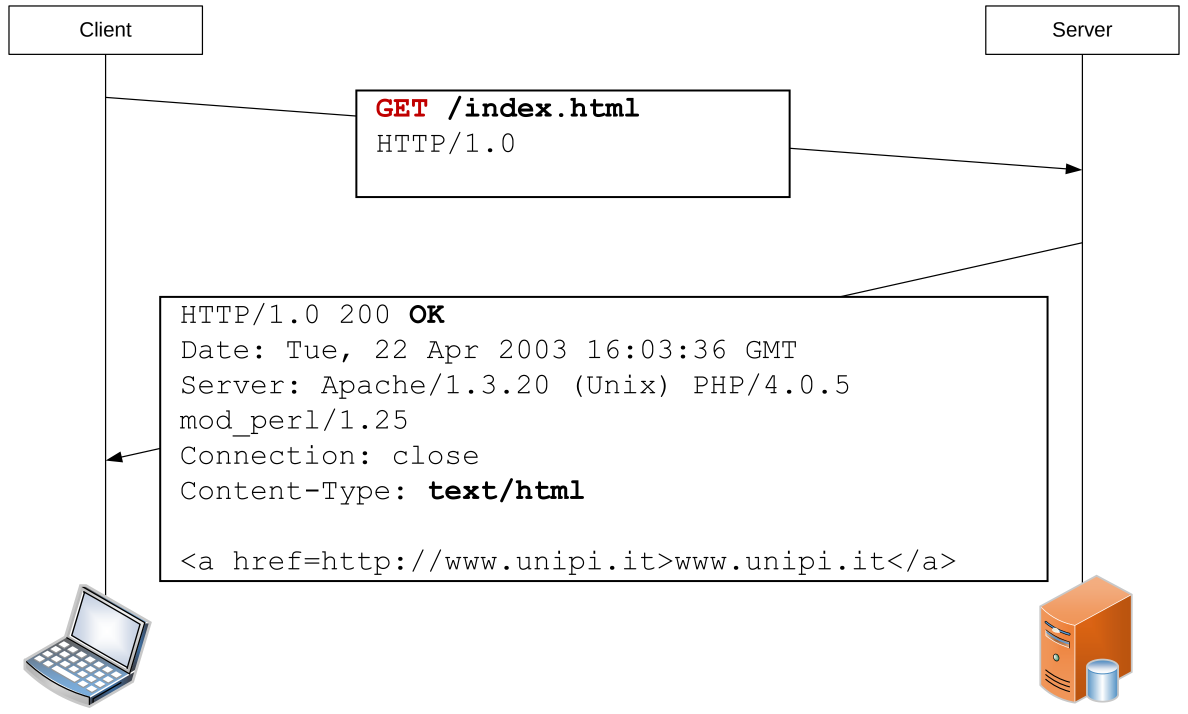
- A platform enabling generic service invocation in distributed computing systems

Uniform Resource Identifier (URI)

- An URI “is a compact string of characters for identifying an abstract or physical resource” [RFC 2396]
- URI syntax [*RFC 1630*]
  - `<uri> ::= <scheme>":"<scheme-specific-part>`
  - valid schemes are managed by IANA
    - `http://www.iana.org/assignments/uri-schemes.html`
- A URI can be
  - A name: Uniform Resource Name (**URN**)
    - `<urn> ::= "urn:" <nid> ":" <nss>`
    - `urn:isbn:0-395-36341-1`
  - An address: Uniform Resource Locator (**URL**)
    - `"http:" "//" <host> [":" <port>] [<abs_path> ["?" query]]`
    - `http://www.unipi.it/`

Hypertext Transfer Protocol (HTTP)

- HTTP "… is an application-level protocol for distributed, collaborative, hypermedia information systems"



```
GET /index.html
HTTP/1.0
```

```
HTTP/1.0 200 OK
Date: Tue, 22 Apr 2003 16:03:36 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
mod_perl/1.25
Connection: close
Content-Type: text/html

<a href=http://www.unipi.it>www.unipi.it</a>
```

Data Encoding

- Web adopts HTML, HyperText Markup Language, that is the perfect solution for visual data presentation, but it is not suited to encode application data
- For this reason, the e**X**tensible **M**arkup **L**anguage or XML has been introduced in 2000 by W3C
- *XML is a Text-based encoding markup language that defines a set of rules for encoding any kind of document in a format that is both human-readable and machine-readable*
- XML specifies the syntax, how information must be written within the document
- Grammar specifications, i.e. what kind of information can be written on document, are also available but not mandatory
- The language is extensible as the tag set is not pre-specified and it can be archived or transmitted via internet easily

XML

- An XML document has a ***hierarchical structure*** and is composed of several elements
- An **element** is a logical document component that begins with a start-tag and ends with an end-tag
- The **document** usually begins with a **root** element, named XML declaration that describes some information about the document itself
- A **tag** is a markup construct that begins with ‘<‘ and ends with ‘>’
- Tags can contain **attributes**, a name-value pair tat can be specified within a start-tag

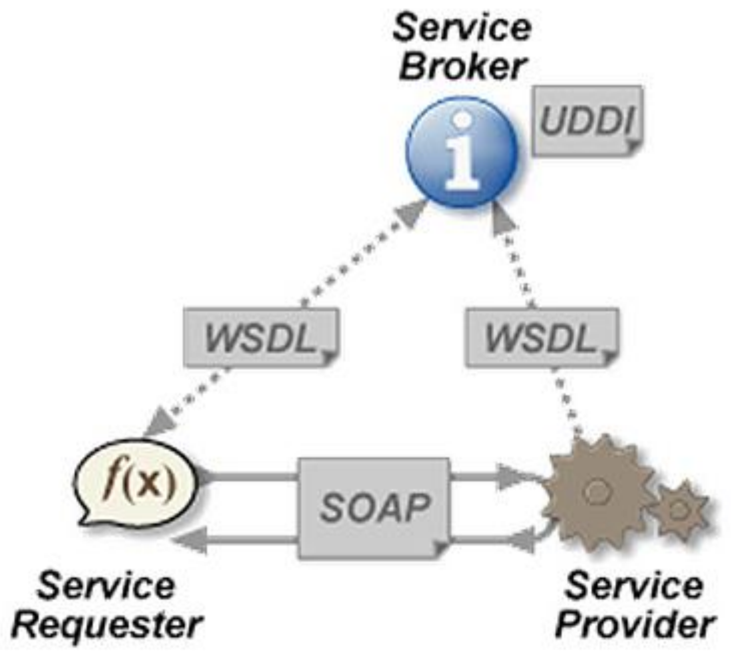
```xml
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>establishment</type>
  <type>point_of_interest</type>
   <geometry>
   <location type='GPS'>
    <lat>43.7210349</lat>
    <lng>10.3898890</lng>
   </location>
  </geometry>
</result>
</GeocodeResponse>
```

XML Validation

- XML syntax specifies if a document is well-formed
- In order to verify if a document is also valid, i.e. it has meaning, it needs to contain a reference to another document, a Document Type Definition (DTD) or an XML Schema (the newer version)
- DTD/XML Schema is a schema or a grammar of the document, i.e. it specifies what kind of information can be included in the document in relation to elements and attributes
- An XML document that is compliant with the rules of a DTD/XML Schema document is marked as valid, non-valid otherwise

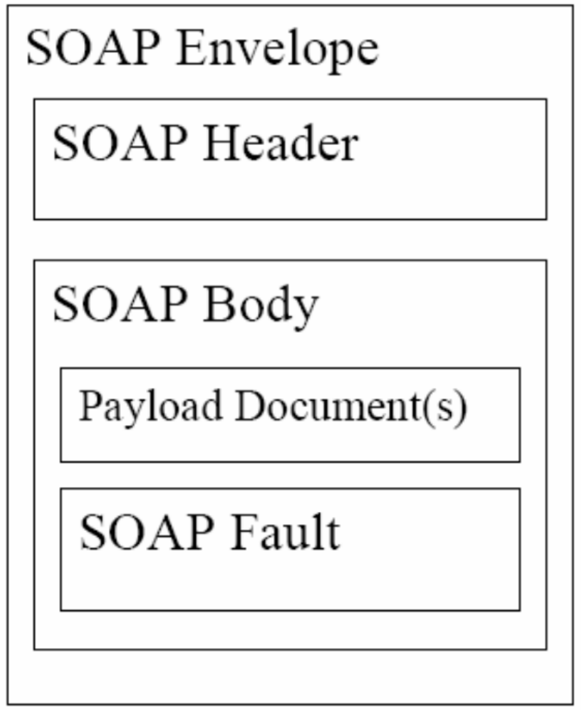
WS - Interaction

- A WS has an interface described in a machine-executable format, the **Web Services Description Language (WSDL)**
- WSDL is used to interact with the broker, a module that provides a global registry for advertising and discovering web services.
- Via WSDL each service provider register itself to the broker, while the service consumer uses WSDL to look up for a specific service
- Service consumer and provider interact using the **Simple Object Access Protocol (SOAP)** that provides a standard messaging format



SOAP

- SOAP provides a standard packaging structure for transmission of XML documents over various internet protocols
- A SOAP Message consists of a root element called **envelope** that contains a **header** and a body
- The header contains additional information and security information
- The body contains the payload, which includes request/response information, and a Fault section that includes errors and exceptions, if any



SOAP Message Example - Request

SOAP Envelope Element

SOAP Body

Name of the WS

URI of the WS

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-
ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
          <soap:Body>
              <MySystemWebService xmlns="http://unipi.it/">
                 <Param1>4</Param1>
              </MySystemWebService>
          </soap:Body>
</SOAP-ENV:Envelope>
```

Parameter for the invocation

SOAP Message Example - Response

SOAP Envelope Element

SOAP Body

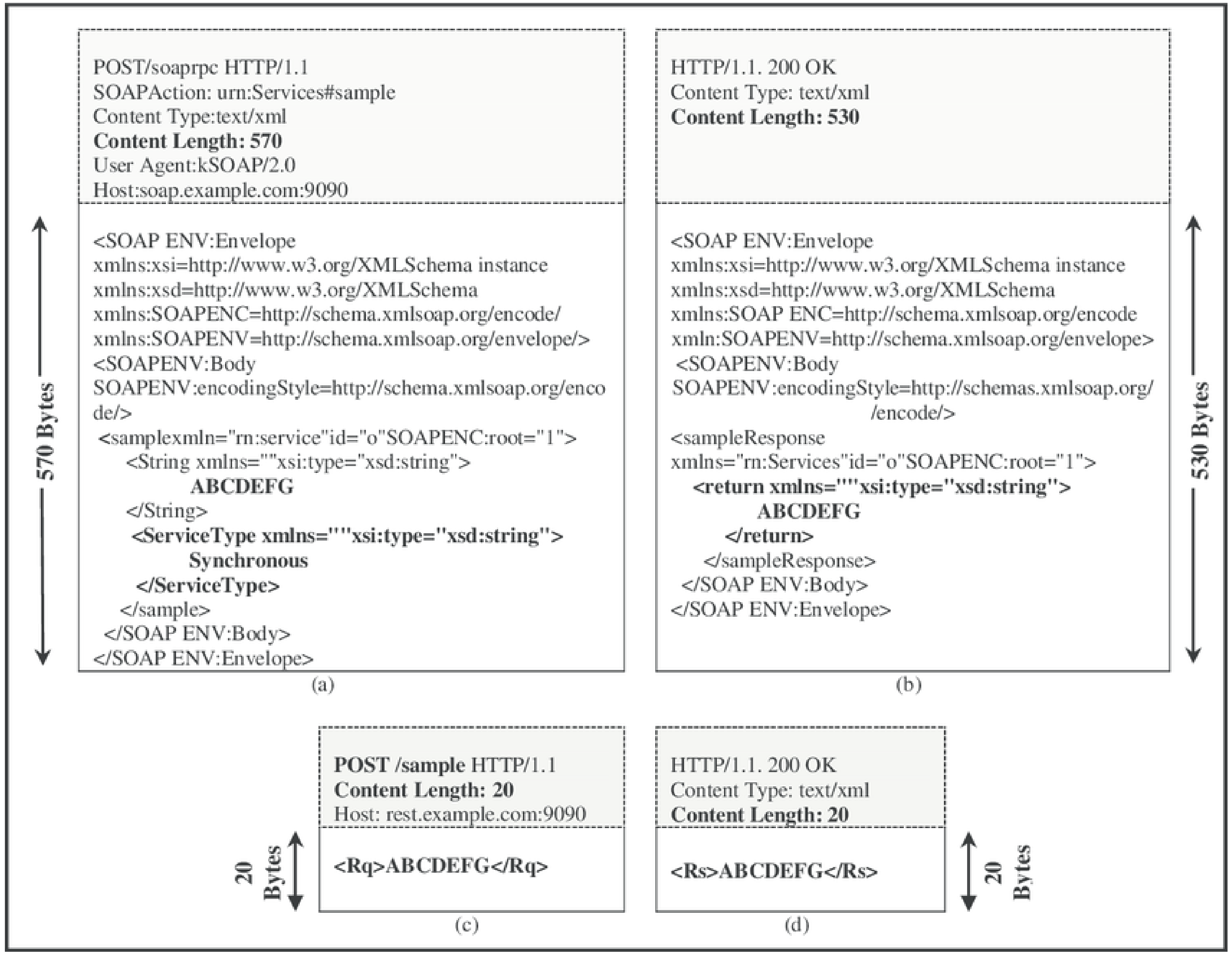
Fault Section

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <SOAP-ENV:Body>
         <SOAP-ENV:Fault>
         <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
         <faultstring xsi:type="xsd:string">
            Failed to locate method
         </faultstring>
     </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Error Message

SOAP Binding

- Typical SOAP binding is HTTP, i.e. it means that SOAP messages are transmitted over HTTP
- SOAP can use GET or POST
- With GET the response is a SOAP message, the request optionally, with POST both request and response are SOAP messages



```
POST/soaprpc HTTP/1.1
SOAPAction: urn:Services#sample
Content Type:text/xml
Content Length: 570
User Agent:kSOAP/2.0
Host:soap.example.com:9090

<SOAP ENV:Envelope
xmlns:xsi=http://www.w3.org/XMLSchema instance
xmlns:xsd=http://www.w3.org/XMLSchema
xmlns:SOAPENC=http://schema.xmlsoap.org/encode/
xmlns:SOAPENV=http://schema.xmlsoap.org/envelope/>
<SOAPENV:Body
SOAPENV:encodingStyle=http://schema.xmlsoap.org/enco
de/>
 <samplexmln="rn:service"id="o"SOAPENC:root="1">
     <String xmlns=""xsi:type="xsd:string">
              ABCDEFG
     </String>
      <ServiceType xmlns=""xsi:type="xsd:string">
              Synchronous
      </ServiceType>
    </sample>
  </SOAP ENV:Body>
</SOAP ENV:Envelope>
```

```
HTTP/1.1. 200 OK
Content Type: text/xml
Content Length: 530

<SOAP ENV:Envelope
xmlns:xsi=http://www.w3.org/XMLSchema instance
xmlns:xsd=http://www.w3.org/XMLSchema
xmlns:SOAP ENC=http://schema.xmlsoap.org/encode
xmln:SOAPENV=http://schema.xmlsoap.org/envelope>
 <SOAPENV:Body
SOAPENV:encodingStyle=http://schemas.xmlsoap.org/
                         /encode/>
<sampleResponse
xmlns="rn:Services"id="o"SOAPENC:root="1">
    <return xmlns=""xsi:type="xsd:string">
           ABCDEFG
        </return>
     </sampleResponse>
  </SOAP ENV:Body>
</SOAP ENV:Envelope>
```

```
POST /sample HTTP/1.1
Content Length: 20
Host: rest.example.com:9090

<Rq>ABCDEFG</Rq>
```

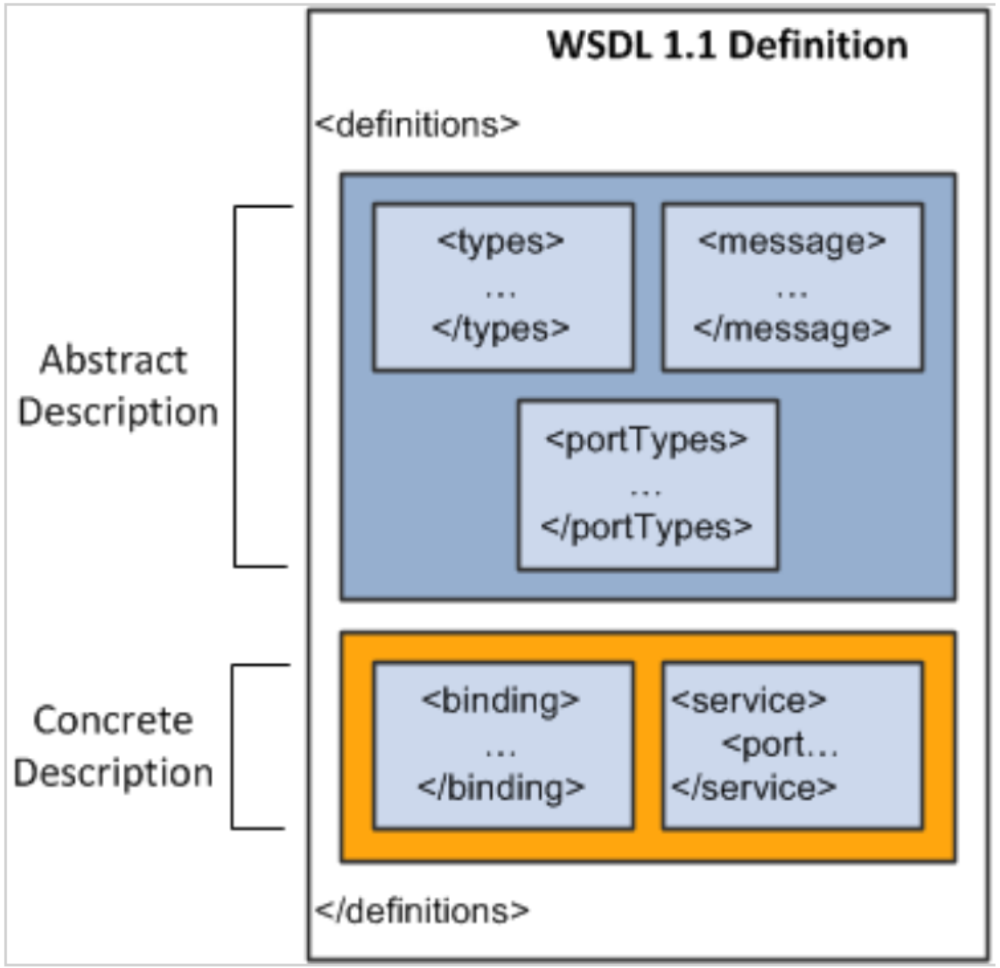
```
HTTP/1.1. 200 OK
Content Type: text/xml
Content Length: 20

<Rs>ABCDEFG</Rs>
```

WSDL

- WSDL describes the interface and the set of operations supported by a WS in a standard format
- It standardizes also the set of input and output parameters of the service operations as well as the service protocol, i.e. the way in which the messages will be transferred to/from the service
- Via WSDL, clients can automatically understand:
  - What a service can do
  - Where the service is located
  - How it can be invoked
- Every time a client wants to invoke an unknown WS, it requests and analyzes the a WSDL document, only afterwards it invokes the service through SOAP

WDSL – Message architecture

- An WDSL message is an XML document and it comprises of two sections:
  - The Service Interface Definition, which defines in an abstract manner the service. It comprises: the type of data (types), the messages exchanged between requestor and provider (messages) and the operations that can be performed (portType)
  - The Service Implementation Definition, which defines in a specific manner how the service is invoked. It comprises: how to transfer the messages (binding) and how to invoke the service (service)



WSDL - Type

Type is used to define complex data types. Such types are then used in the messages.

```
<types>
    <element name=“DataIDRequest">
        <complexType>
             <all>
                   <element name=“DataID" type="string"/>
             </all>
        </complexType>
    </element>
</types>
```

Name of the type

The type is a data structure (complex)

Name of the field and type (string)

WSDL - Messages

Message is used to define the message that is exchanged between client and WS. The message can define both an input or output message.

Name of the message

Fields of the message

```
<message name=“GetDataByIDInput">
      <part name="body" element="xsd1:DataIDRequest"/>
</message>
<message name="GetDataByIDOutput">
      <part name="body" element="xsd1:Data"/>
</message>
```

The response contains the data

The message contains one field, the ID of the data requested

WSDL - PortType

PortType is used to link input and output messages into one logical operation.

Name of the PortType

Name of the operation

```
<portType name=“GetDataPortType">
    <operation name=“GetDataName">
       <input message="tns:GetDataByIDInput"/>
       <output message="tns:GetDataByIDOutput"/>
    </operation>
</portType>
```

Input and output messages

WSDL - Binding

Binding is used specify how the messages associated to one PortType operation are exchanged

```
<binding name="GetDataBinding" type="tns:GetDataPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetDataName">
              <soap:operation soapAction="http://unipi.it/getdataname"/>
              <input>
                <soap:body use="literal"/>
              </input>
              <output>
                <soap:body use="literal"/>
              </output>
        </operation>
 </binding>
```

The name of the PortType

The SOAP Action parameters

WSDL - Service

Service is used to specify how the operation is practically invoked, i.e. the URI associated with the service

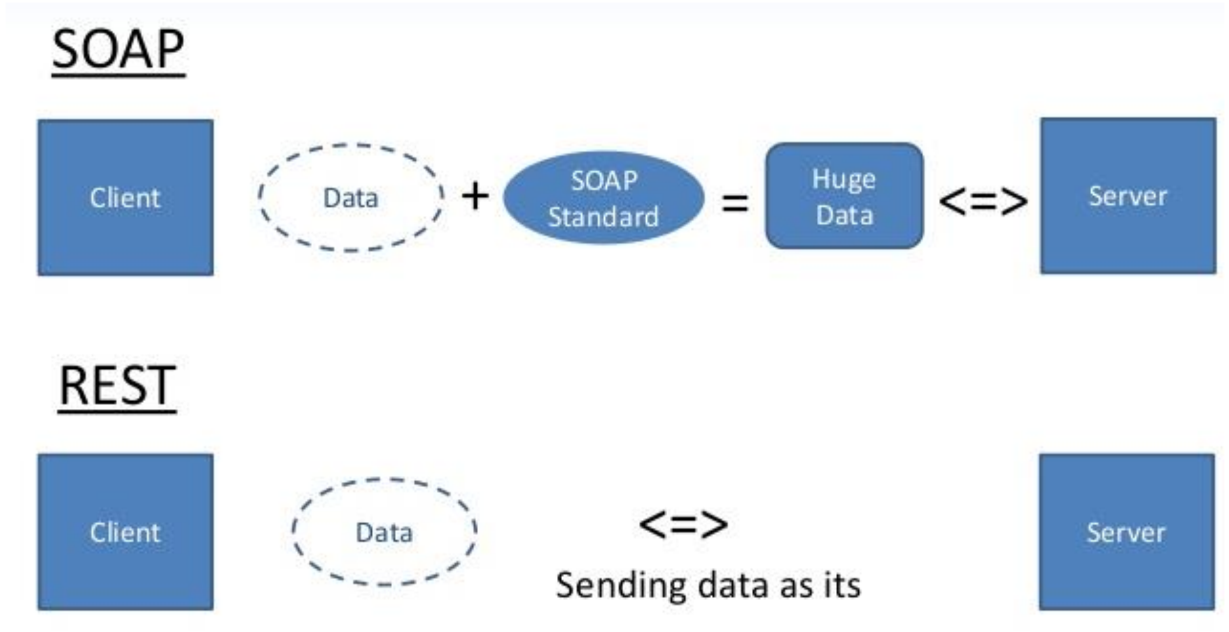
```
<service name=“GetDataService">
    <documentation>GetDataService</documentation>
    <port name=“GetData" binding="tns:GetDataBinding">
      <soap:address location="http://unipi.it/getdata"/>
    </port>
 </service>
```

The binding associated

The URI for the SOAP invocation

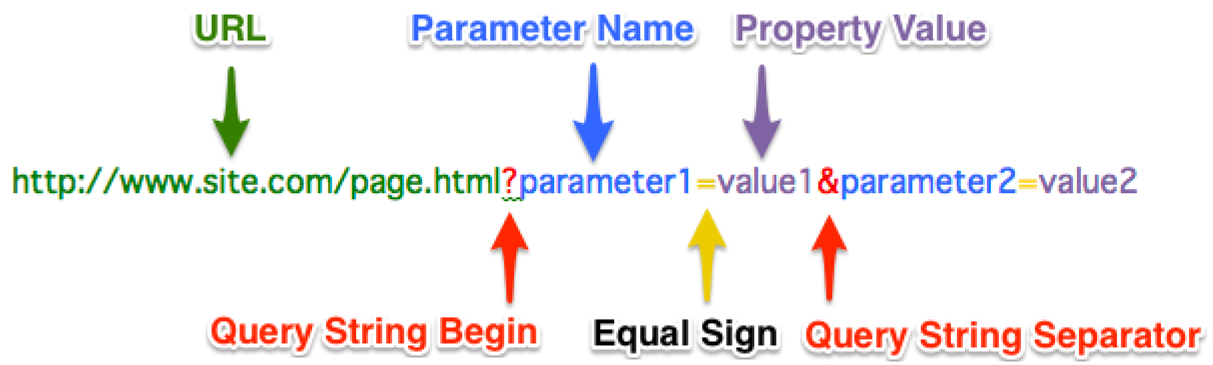
REST

- SOAP is often considered as complex protocol, as it requires the creation of the XML structure, which is mandatory for passing messages
- *The Representational State Transfer (REST) has been introduced as valid lightweight alternative for WS construction*
- It provides a model for designing network-based software systems utilizing the client/server model based on the HTTP as transfer protocol
- In a RESTful system, a client sends a request over HTTP using the standard HTTP methods (PUT, GET, POST, and DELETE), and the server issues a response that includes the representation of the resource
- By relying on the minimal support offered by HTTP, it is possible to provide whatever it is needed to replace the basic and most important functionality provided by SOAP: **the method invocation**
- The content of data is still transmitted using XML as part of the HTTP content, but the additional markup required by SOAP is removed

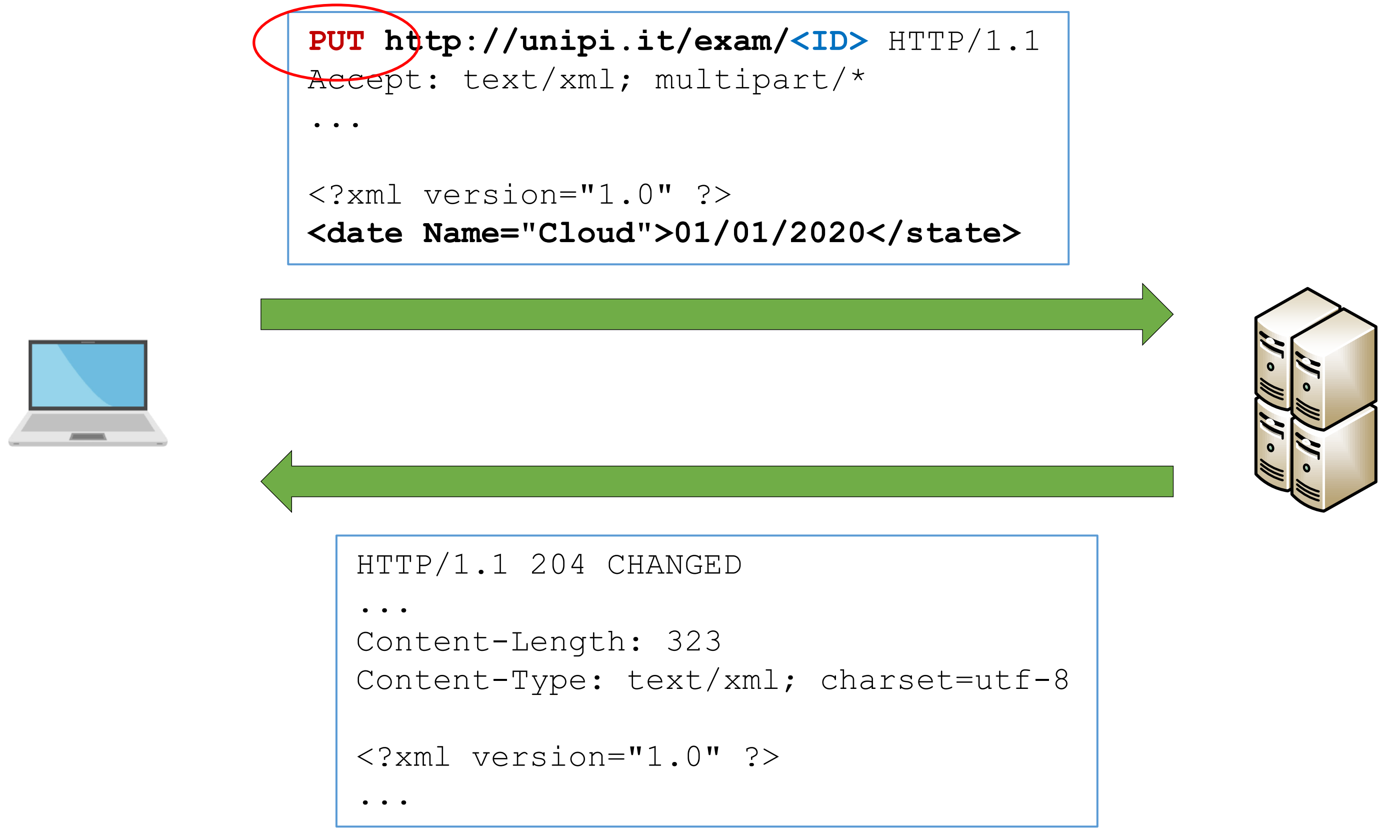


REST

- The GET, PUT, POST, and DELETE methods constitute a minimal set of operations for retrieving, adding, modifying, and deleting data
- Parameters to be included in the request, or data to be sent to the service could be included as query values in the request (parameters) or as payload (data to be uploaded)
- Together with an appropriate URI organization to identify resources, all the atomic operations required by a WS can be implemented
- For this reason, REST represents a lightweight alternative to SOAP, which works effectively in contexts where additional aspects beyond those manageable through HTTP are absent
- The global identifier assigned to each resource makes the access method of the resources uniform. Any resource at a known address can be accessed by any application agents.
- REST allows many standard formats like XML, JavaScript Object Notation (JSON) or plain text as well as any other agreed upon formats for data exchange



REST - Example



```
PUT http://unipi.it/exam/<ID> HTTP/1.1
Accept: text/xml; multipart/*
...

<?xml version="1.0" ?>
<date Name="Cloud">01/01/2020</state>
```

```
HTTP/1.1 204 CHANGED
...
Content-Length: 323
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" ?>
...
```

JSON

- JavaScript Object Notation (JSON) is an open standard format that uses human-readable text to transmit data objects
- It was originally defined for web applications and asynchronous data interchange (AJAX)
- It is based on a subset of JavaScript and is a text format and language independent
- It is designed to be a lightweight data encoding, based on only two structures
    - Collection of name/value pairs
    - Ordered list of values

JSON vs XML

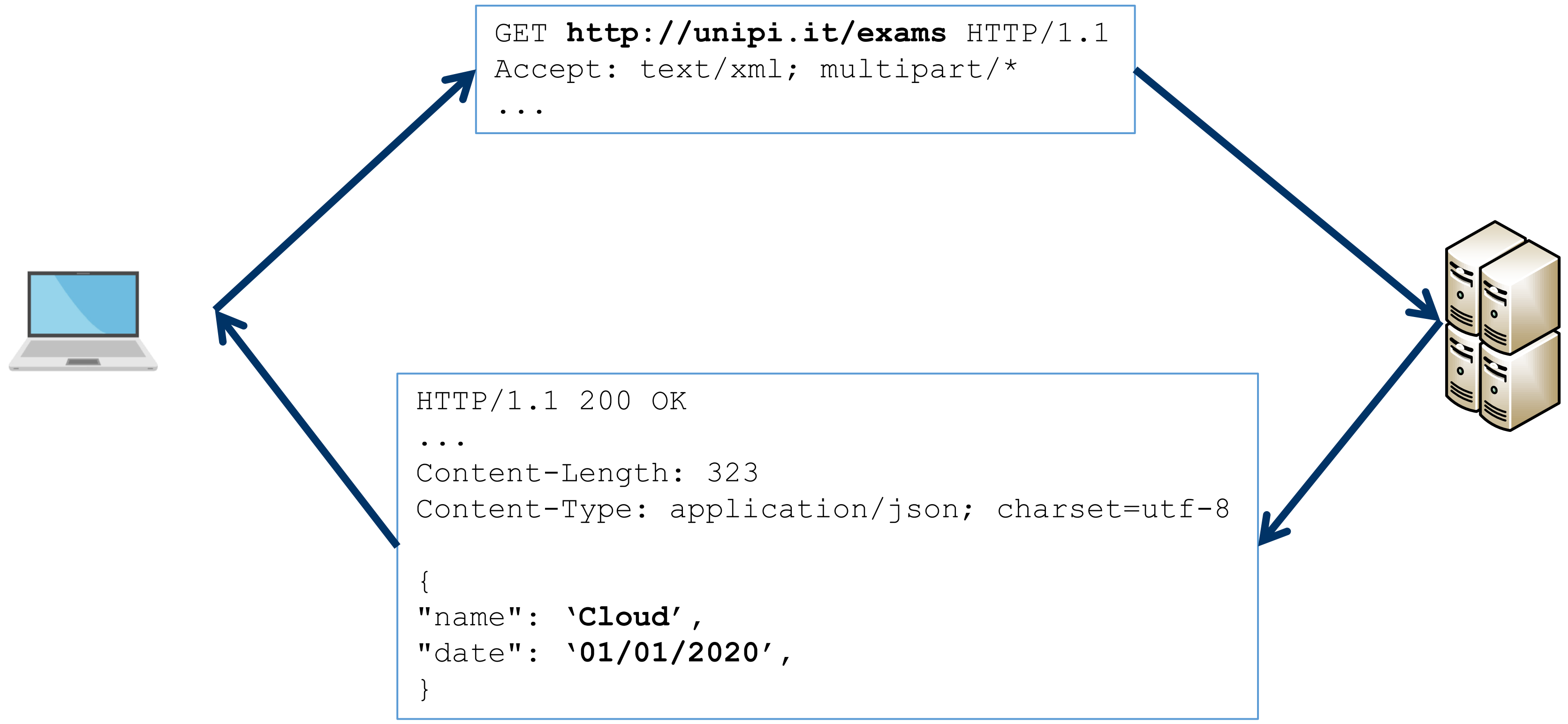
JSON

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

JSON in WS



```
GET http://unipi.it/exams HTTP/1.1
Accept: text/xml; multipart/*
...
```

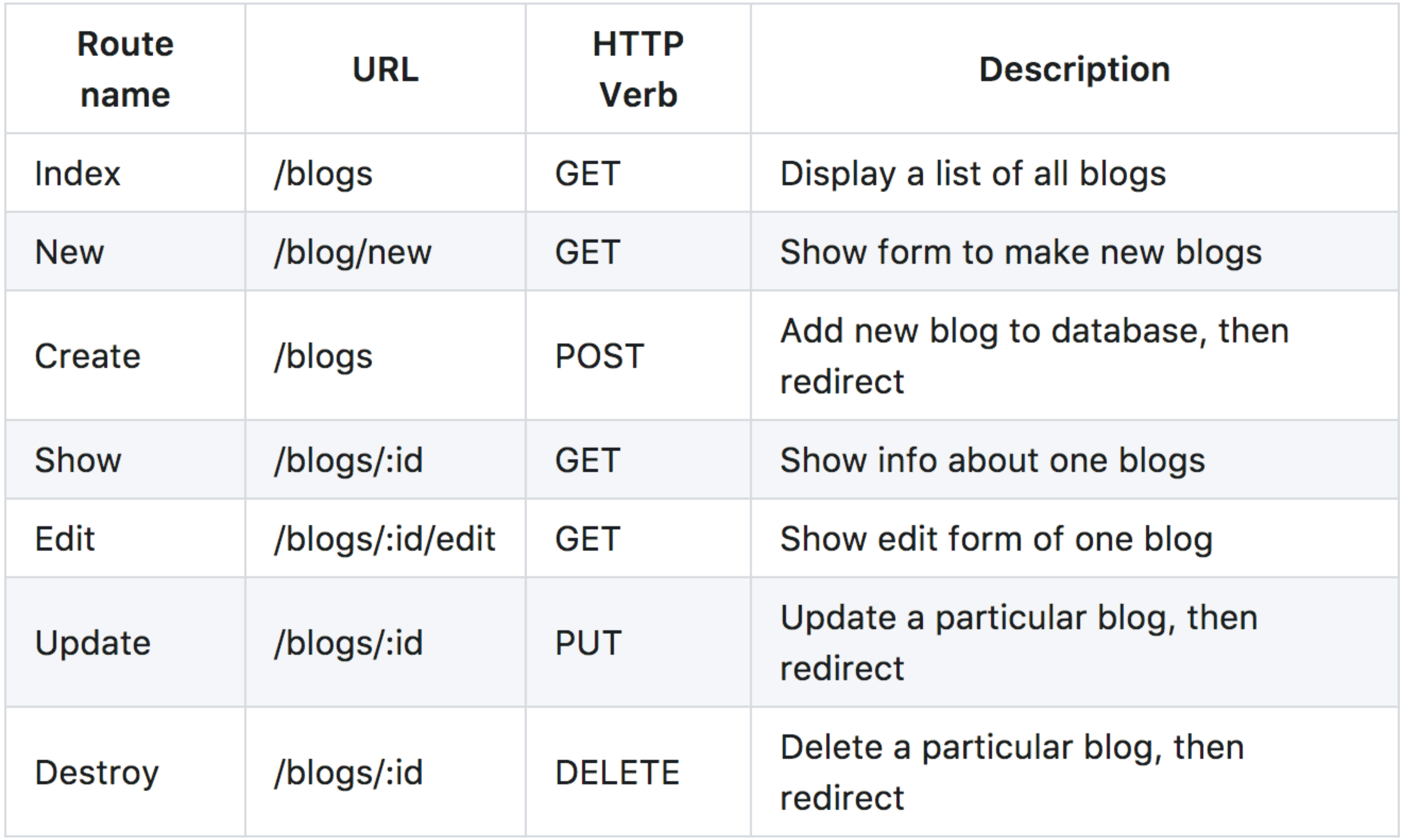
```
HTTP/1.1 200 OK
...
Content-Length: 323
Content-Type: application/json; charset=utf-8

{
"name": ‘Cloud’,
"date": ‘01/01/2020’,
}
```

CRUD

- The acronym CRUD refers to all of the major functions that are implemented in relational database applications: Create, Read, Update and Delete
- Each letter in the acronym can map to a standard Structured Query Language (SQL) statement, Hypertext Transfer Protocol (HTTP) method
- Create
  - CREATE procedures generate new records via INSERT statements.
- Read/Retrieve
  - READ procedures reads the data based on input parameters. Similarly, RETRIEVE procedures grab records based on input parameters.
- Update
  - UPDATE procedures modify records without overwriting them.
- Delete
  - DELETE procedures delete where specified.

CRUD

- This is typically used to build RESTful APIs
- CRUD principles are mapped to REST commands to comply with the goals of RESTful architecture



Route name	URL	HTTP Verb	Description
Index	/blogs	GET	Display a list of all blogs
New	/blog/new	GET	Show form to make new blogs
Create	/blogs	POST	Add new blog to database, then redirect
Show	/blogs/:id	GET	Show info about one blogs
Edit	/blogs/:id/edit	GET	Show edit form of one blog
Update	/blogs/:id	PUT	Update a particular blog, then redirect
Destroy	/blogs/:id	DELETE	Delete a particular blog, then redirect