

# 8th Slide Set Cloud Computing

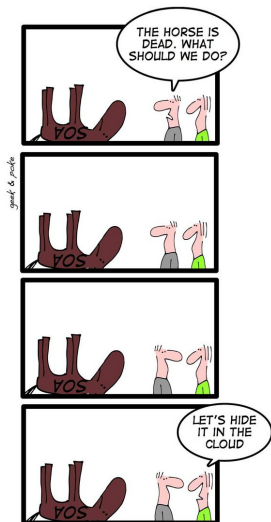
Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Faculty of Computer Science and Engineering  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

# Agenda for Today

- Service-oriented architectures
- Web services
  - Big web services
    - SOAP
    - WSDL
    - UDDI
    - WS-Inspection
  - Web services implemented with REST

# Service-oriented Architectures (SOA)



- SOA is a method to encapsulate IT components in services and coordinate (*orchestrate*) them
- Objective: Grouping of services to higher-level services and simple provisioning to other organizational departments or customers
- Theory: Software development costs can be reduced in the long term
  - Reason: In the future, eventually, all required services are available and they only need to be orchestrated
- Technical implementation: **Web services**

# Web Services

- Distributed systems usually integrate heterogeneous resources
  - In theory, the resources can be distributed worldwide
- Connections via long distances have fundamental disadvantages, compared with LANs
  - High response times
  - Low data rates
  - Potentially unreliable connections
- In such environments, the **weakly coupled, asynchronous** and **message-based** communication via web services is ideal
- Web services can be described as a distributed middleware, which allows machine-to-machine communication, based on web protocols



## Remote Procedure Calls via XML-RPC

- XML-RPC is a simple protocol for remote procedure calls via HTTP
  - XML is used to specify the input parameters for the remote procedures in a well-defined structure
  - XML-RPC is language independent
    - Servers and clients can be implemented in any programming language, e.g. Java, Python, Ruby, C/C++ or Perl
  - Function calls are transmitted via HTTP POST to the server
    - The server evaluates the XML document, which is contained in the message, and uses its content as parameter for calling the desired function
    - The result is written into a XML document and transmitted to the client
- XML-RPC can be seen as the predecessor of SOAP
  - It provides a much lesser functionality, but it is easier to understand

## XML-RPC Example (Server written in Perl)

- Objective: A function, which adds and subtracts

```

1  #!/usr/bin/perl
2  use strict;
3  use Frontier::Daemon;
4
5  sub sumAndDifference {
6      my ($x, $y) = @_;
7      return {'sum' => $x + $y, 'difference' => $x - $y};
8  }
9
10 # Call me as http://localhost:8080/RPC2
11 my $methods = {'sample.sumAndDifference' => \&sumAndDifference};
12 Frontier::Daemon->new(LocalPort => 8080, methods => $methods)
13     or die "Couldn't start HTTP server: $!";

```

Source: <http://cseweb.ucsd.edu/classes/su10/cse120/lectures/xmlrpc.pl>

# XML-RPC Example (Client written in Perl)

```

1 #!/usr/bin/perl
2 use strict;
3 use Frontier::Client;
4
5 unless (scalar(@ARGV) == 2) {
6     die "usage: $0 x y\n";
7 }
8
9 # Make an object to represent the XML-RPC server.
10 my $server_url = 'http://localhost:8080/RPC2';
11 my $server = Frontier::Client->new(url => $server_url);
12
13 # Call the remote server and get our result.
14 my $result = $server->call('sample.sumAndDifference', @ARGV);
15 my $sum = $result->{'sum'};
16 my $difference = $result->{'difference'};
17
18 print "Sum: $sum, Difference: $difference\n";

```

Source: <http://cseweb.ucsd.edu/classes/su10/cse120/lectures/xmlrpc.pl>



# XML-RPC Example (Request via HTTP POST)

```
1 POST /RPC2 HTTP/1.1
2 TE: deflate,gzip;q=0.3
3 Connection: TE, close
4 Host: localhost:8080
5 User-Agent: libwww-perl/5.808
6 Content-Type: text/xml
7 Content-Length: 199
8
9 <?xml version="1.0"?>
10 <methodCall>
11   <methodName>sample.sumAndDifference</methodName>
12   <params>
13     <param>
14       <value><int>2</int></value>
15     </param>
16     <param>
17       <value><int>3</int></value>
18     </param>
19   </params>
20 </methodCall>
```

Source: <http://cseweb.ucsd.edu/classes/su10/cse120/lectures/xmlrpc.pl>

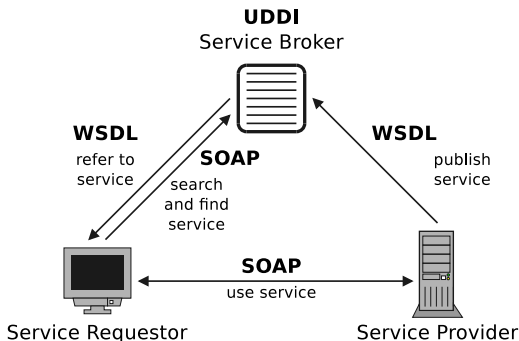
# XML-RPC Example (Reply)

```
1 HTTP/1.1 200 OK
2 Date: Tue, 27 Jul 2010 18:19:10 GMT
3 Server: libwww-perl-daemon/1.39
4 Content-Length: 252
5 Content-Type: text/xml
6
7 <?xml version="1.0"?>
8 <methodResponse>
9   <params>
10     <param>
11       <value>
12         <struct>
13           <member>
14             <name>difference</name>
15             <value><int>-1</int></value>
16           </member>
17           <member>
18             <name>sum</name>
19             <value><int>5</int></value>
20           </member>
21         </struct>
22       </value>
23     </param>
24   </params>
25 </methodResponse>
```

Source: <http://cseweb.ucsd.edu/classes/su10/cse120/lectures/xmlrpc.pl>

# Big Web Services with SOAP in Theory

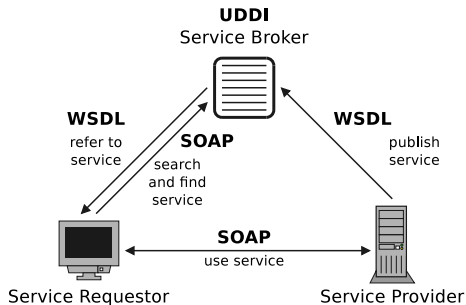
- Web services with SOAP are also called big web services
- The theoretical implementation includes 3 roles
  - Service provider
  - Service requestor (user / customer)
  - Service registry (broker)



# Web Services with SOAP: Components

- The providers register their services in a registry
- Users search for matching services in the registry (= broker)
- If a matching service is found, the user receives a reference (address) from the registry to the interface description (WSDL)
  - The user can call the web service with the provided information

- Function calls and data exchange is carried out via the network protocol SOAP
- Registry, WSDL descriptions and SOAP messages use the markup language XML

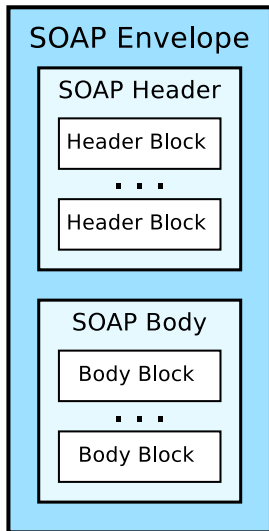


# Web Services: SOAP

- The SOAP XML vocabulary is the basis of traditional web services
- Usually, SOAP messages are sent to a URL in the payload section (*body*) of a HTTP POST request
  - Other protocols, such as SMTP (asynchronous) or FTP (large amount of data) are also possible
- SOAP was created in 1999
  - Successor of XML-RPC
  - Originally defined as Simple Object Access Protocol (SOAP)

Since version 1.2, the abbreviation SOAP is no longer used as an acronym, because it is (subjectively) not **S**imple to use/understand and it is not only used for **O**bject **A**ccess

# Web Services: SOAP



- SOAP messages are XML documents, which contain a **SOAP envelope** with 2 message parts
- The optional **SOAP header** may contain among others, information for routing, authentication and authorization
  - The SOAP header contains no payload data
- The absolutely required **SOAP body** with the message („payload“) may contain informations for data exchange or instructions for a remote procedure call

## Example for a XML-based SOAP Message

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
3   <env:Header>
4     <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
5       <n:priority>1</n:priority>
6       <n:expires>2001-06-22T14:00:00-05:00</n:expires>
7     </n:alertcontrol>
8   </env:Header>
9   <env:Body>
10    <m:alert xmlns:m="http://example.org/alert">
11      <m:msg>Pick up Mary at school at 2pm</m:msg>
12    </m:alert>
13  </env:Body>
14 </env:Envelope>
```

Source: Tanenbaum, van Stee. Distributed Systems: Principles and Paradigms. Prentice Hall (2006)

- The XML message transmits a text to a web service
- The message contains a priority value (1) and is discarded, if it arrives after 14:00 o'clock at the web service

# Web Services: WSDL

- WSDL (Web Services Description Language) is an XML-based interface description language to specify...
  - how a web service can be called
  - what parameters a web service expects
  - what data structures a web service returns
- A WSDL description contains:
  - Description of the data types used by the web service
  - Description of the operations, provided by the web service
  - Protocol descriptions
  - Interfaces and ports



## WSDL Example – Part 1/5

- This WSDL example is from the lecture *Web Services* from Ingo Melzer (WS0304)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3   targetNamespace="http://localhost/axis/Calc.jws"
4   xmlns="http://schemas.xmlsoap.org/wsdl/"
5   xmlns:apachesoap="http://xml.apache.org/xml-soap"
6   xmlns:impl="http://localhost/axis/Calc.jws"
7   xmlns:intf="http://localhost/axis/Calc.jws"
8   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
9   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
10  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
11  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12  <!-- portType -->
13  <!-- message -->
14  <!-- binding -->
15  <!-- service -->
16 </wsdl:definitions>
```

- The root element `<definitions>` with the required namespaces

## WSDL Example – Part 2/5

```
1 <!-- definitions -->
2 <wsdl:portType name="Calc">
3   <wsdl:operation name="add" parameterOrder="a b">
4     <wsdl:input message="impl:addRequest" name="addRequest"/>
5     <wsdl:output message="impl:addResponse" name="addResponse"/>
6   </wsdl:operation>
7 </wsdl:portType>
8 <!-- message -->
9 <!-- binding -->
10 <!-- service -->
```

- Specification of the add operation for adding 2 values
- The web service can receive a message (addRequest) and sends a response (addResponse)

# WSDL Example – Part 3/5

```
1 <!-- definitions -->
2 <!-- portType -->
3 <wsdl:message name="addRequest">
4   <wsdl:part name="a" type="xsd:int"/>
5   <wsdl:part name="b" type="xsd:int"/>
6 </wsdl:message>
7 <wsdl:message name="addResponse">
8   <wsdl:part name="addReturn" type="xsd:int"/>
9 </wsdl:message>
10 <!-- binding -->
11 <!-- service -->
```

- Specification of input and output parameters (in this example, all integers)
- The request (addRequest) contains 2 parameters (a and b)
- The response (addResponse) contains 1 parameter (addReturn)

# WSDL Example – Part 4/5

```
1 <!-- definitions -->
2 <!-- portType -->
3 <!-- message -->
4 <wsdl:binding name="CalcSoapBinding" type="impl:Calc">
5   <wsdlsoap:binding style="rpc"
6     transport="http://schemas.xmlsoap.org/soap/http"/>
7   <wsdl:operation name="add">
8     <wsdlsoap:operation soapAction=""/>
9     <wsdl:input name="addRequest">
10      <wsdlsoap:body
11        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
12        namespace="http://DefaultNamespace" use="encoded"/>
13    </wsdl:input>
14    <wsdl:output name="addResponse">
15      <wsdlsoap:body
16        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
17        namespace="http://localhost/axis/Calc.jws" use="encoded"/>
18    </wsdl:output>
19  </wsdl:operation>
20 </wsdl:binding>
21 <!-- service -->
```

- The web service is addressed in RPC style via HTTP
- Protocols for input and output (addRequest and addResponse) of the add method are referenced and therefore specified with encodingStyle

# WSDL Example – Part 5/5

```
1 <!-- definitions -->
2 <!-- portType -->
3 <!-- message -->
4 <!-- binding -->
5 <wsdl:service name="CalcService">
6   <wsdl:port binding="impl:CalcSoapBinding" name="Calc">
7     <wsdlsoap:address location="http://localhost/axis/Calc.jws"/>
8   </wsdl:port>
9 </wsdl:service>
```

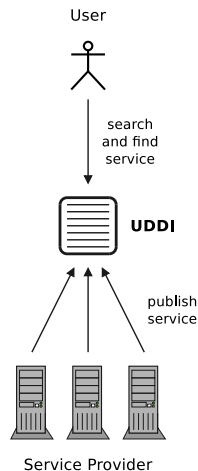
- Here, the definition of the port(s) to access the web service takes place
- Multiple access points can be specified per web service
- The web service runs locally and is located in /axis/Calc.jws

# Web Services: UDDI

- UDDI is a directory service
  - UDDI = Universal Description, Discovery and Integration
  - Web services are classified, cataloged and managed in UDDI directories
  - Provides a standardized directory structure for the metadata of web services
    - Technical characteristics, requirements and provider information
  - Based on XML
- A UDDI service may contain 3 types of information:
  - ① White Pages
    - Similar to a telephone book
    - Contains the address and contact information of the service provider
  - ② Yellow Pages
    - Similar to a business directory/mercantile directory (⇒ Yellow Pages)
    - Categorizations of the services, according to their purpose
  - ③ Green Pages
    - Contains technical information about services
    - Includes the interface descriptions

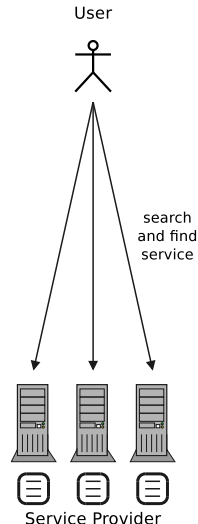
# Architecture and Data Model of UDDI

- Data model (data structures)
  - `businessEntity`: Information about the provider
  - `businessService`: General description of the service
  - `bindingTemplate`: Technical characteristics of the service, which are required to use the service (e.g. URI)
  - `tModel`: Technical specification of the service
  - `publisherAssertion`: Describes the relationship between multiple service providers
    - Possible scenario: A company has multiple divisions
  - `operationalInfo`: Stores modifications in the UDDI registry
    - If a record in the registry is modified, here it is specified, who caused the modification and on which UDDI node the record was created or modified



# Alternative to UDDI: WS-Inspection

- More simple concept to locate web services compared with UDDI
- Different Architectures
  - UDDI = few, centralized directories, where different providers publish their services
  - WS-Inspection = many decentralized, small directories, in which few providers publish their services
- WS-Inspection bypasses 2 issues of UDDI:
  - The lack of adequate moderation of the directories cause poor search results
  - WS-Inspection supports searching for services only at providers, the customers trust
- WS-Inspection documents are published in the root directory of the web server of the provider





# WS-Inspection

- IBM and Microsoft specified in 2001 how to find WS-Inspection documents

<http://www-128.ibm.com/developerworks/library/specification/ws-wsilspec/>

- WS-Inspection is entirely document based
  - At the web server of the provider, a document with a predefined name `inspection.wsil` is stored
  - The document contains information about the offered services
  - Users request the document via HTTP and receive a list of web services and a description in the WSDL format
- With the `link` element, a hierarchy of WS-Inspection documents can be realized
  - A WS-Inspection document refers via `link` to one or more subordinate WS-Inspection documents
- WS-Inspection did not gain wide acceptance

# Example of a WS-Inspection Document

```
1 <?xml version="1.0"?>
2 <inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
3   <service>
4     <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
5       location="http://test.uddi.microsoft.com/inquire.asmx?WSDL" />
6   </service>
7   <service>
8     <description referencedNamespace="urn:uddi-org:api">
9       <wsiluddi:serviceDescription location="http://uddi.ibm.com/ubr/publishapi">
10        <wsiluddi:serviceKey>52946BB0-BC28-11D5-A432-0004AC49CC1E</wsiluddi:serviceKey>
11      </wsiluddi:serviceDescription>
12    </description>
13  </service>
14  <link referencedNamespace="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
15    location="http://wetter.fmi.uni-passau.de/inspection.wsil" />
16 </inspection>
```

- 2 services are provided:
  - The 1st service refers to the WSDL description of the inquiry API of the Microsoft UDDI test registry
  - The 2nd service is the publishing API of an UDDI node from IBM
- Additionally, a link to another WS-Inspection document is contained

Source: Service-orientierte Architekturen mit Web Services. Elsevier (2005)

# Web Services with SOAP in Practice

- The structure of a web service environment is in practice today usually different
- Publicly accessible web services are usually offered without using UDDI
- Reason:
  - The individual web services and the required access information are already known to the user or client application
  - For this reason, the users and client applications no longer need to search for a web services in a registry service
- Effect:
  - On 12 January 2006, the UDDI Business Registry (UBR), which was operated by IBM, Microsoft and SAP and was the largest UDDI directory at that time, was shut down
- Equal to WS-Inspection, UDDI did not gain wide acceptance

# Web Services with SOAP in Practice

- Today, web services operate in practice according to the illustrated scenario



- Only service user and service provider are involved
- Users know the services, they want to use, their accessibility and the providers
- An external registry service is not necessary

# Web Services with REST

- Communication via RESTful web services is carried out only via HTTP
  - REST implements stateless communication
    - The server does not store any state information about the client
- SOAP uses XML for requests
- With REST, a transfer of a state representation takes place
  - REST = REpresentational State Transfer
- The 4 HTTP methods PUT, GET, POST and DELETE are sufficient to initiate all necessary functions on objects
  - Create record
  - Read record
  - Update record
  - Erase record

PhD thesis of Roy Thomas Fielding

*Architectural Styles and the Design of Network-based Software Architectures*. 2000

# HTTP Methods with REST Web Services

<b>HTTP</b>	<b>CRUD Actions</b>	<b>SQL</b>	<b>Description</b>
PUT/POST	Create	INSERT	Create or replace a resource
GET	Read/Retrieve	SELECT	Request a resource
PUT	Update	UPDATE	Modify a resource
DELETE	Delete/Destroy	DELETE	Erase a resource

- The methods are similar to the CRUD actions from the database field
  - CRUD represents the basic operations on permanent (persistent) memory, which are Create, Read/Retrieve, Update and Delete/Destroy
- For comparison, the table contains the matching SQL statements too

# HTTP Methods with REST Web Services

- In addition to the 4 HTTP methods, the methods HEAD and OPTIONS exist
  - HEAD – requests metadata about a resource (file) from the server
    - This way, metadata of a resource can be requested, without transferring the resource itself
    - The same header is returned as with GET
  - OPTIONS – requests, which methods are supported by a resource
- Examples:
  - If from the object-based storage service S3 a resource, in this case the object `example.txt`, shall be requested, this is carried out via the HTTP method GET  
`GET /example.txt HTTP/1.1`
  - Erasing the resource (the object) is done under the condition that the user privileges allow this action, via the HTTP method DELETE  
`DELETE /example.txt HTTP/1.1`

# Request a List of Buckets via REST

- Request from the client

```
1 GET / HTTP/1.1
2 Host: s3.amazonaws.com
3 Accept-Encoding: identity
4 Date: Fri, 29 Oct 2010 11:05:23 GMT
5 Content-Length: 0
6 Authorization: AWS 1QHSHZGENHP57C0JK5JSZEBAS23AS...usw...
7 User-Agent: Boto/1.8d (linux2)
```

- First part of the authorization: AWS Access Key
- Second part of the authorization: Signature
  - Structure of the signature  $\implies$  slide set 5



# List of Buckets: Response of from S3 ( $\implies$ 2 Buckets)

```

1 HTTP/1.1 200 OK
2 x-amz-id-2: JaYLRR6nqyUpOM+eWegCmaBNC2NZJZ...etc...
3 x-amz-request-id: 1FCDF412571D35AA
4 Date: Fri, 29 Oct 2010 11:05:25 GMT
5 Content-Type: application/xml
6 Transfer-Encoding: chunked
7 Server: AmazonS3
8
9 <?xml version="1.0" encoding="UTF-8"?>
10 <ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
11   <Owner>
12     <ID>af0af9137ff66f0aeceb68dd1e50a91...usw...</ID>
13     <DisplayName>christianbaun</DisplayName>
14   </Owner>
15   <Buckets>
16     <Bucket>
17       <Name>christianbaun</Name>
18       <CreationDate>2010-08-10T14:20:10.000Z</CreationDate>
19     </Bucket>
20     <Bucket>
21       <Name>cloud-vorlesung-bucket</Name>
22       <CreationDate>2010-10-28T21:52:46.000Z</CreationDate>
23     </Bucket>
24   </Buckets>
25 </ListAllMyBucketsResult>

```

The HTTP headers `x-amz-id-2` and `x-amz-request-id` are automatically generated new by Amazon on every request and appended to the response to trace issues

Source: <http://developer.amazonwebservices.com/connect/thread.jspa?messageID=129503>

# Request the List of Keys inside a Bucket via REST

- Request from the client

```
1 GET /cloud-vorlesung-bucket/ HTTP/1.1
2 Host: s3.amazonaws.com
3 Accept-Encoding: identity
4 Date: Fri, 29 Oct 2010 11:42:09 GMT
5 Content-Length: 0
6 Authorization: AWS AKIAI4QECP523R4SGJJJASHEU27HAZW...usw...
7 User-Agent: Boto/1.8d (linux2)
```

# List of Keys inside a Bucket: Response from S3

```
1 HTTP/1.1 200 OK
2 x-amz-id-2: K3xUp0C3VkJG0K2Ix9wgDRPLCysFw...usw...
3 x-amz-request-id: D25FD87E92C49520
4 Date: Fri, 29 Oct 2010 11:42:10 GMT
5 Content-Type: application/xml
6 Transfer-Encoding: chunked
7 Server: AmazonS3
8
9 <?xml version="1.0" encoding="UTF-8"?>
10 <ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
11   <Name>cloud-vorlesung-bucket</Name>
12   <Prefix></Prefix>
13   <Marker></Marker>
14   <MaxKeys>1000</MaxKeys>
15   <IsTruncated>>false</IsTruncated>
16   <Contents>
17     <Key>Testdatei.txt</Key>
18     <LastModified>2010-10-28T21:54:11.000Z</LastModified>
19     <ETag>"20437f481bb3f606f6c4bb64e997da7d"</ETag>
20     <Size>44</Size>
21     <Owner>
22       <ID>af0af9137ff66f0aeeeb68dd1e50a91...usw...</ID>
23       <DisplayName>christianbaun</DisplayName>
24     </Owner>
25     <StorageClass>STANDARD</StorageClass>
26   </Contents>
27
28 ...
29
30 </ListBucketResult>
```

# Request the List of Availability Zones in EC2 via REST

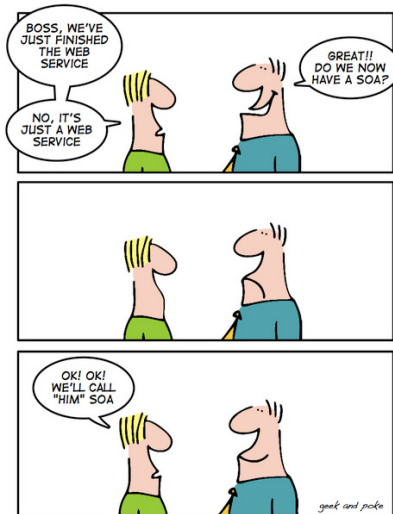
- Request from the client

```
1 GET /?AWSAccessKeyId=AKIAI4QECP523R4SGJJQ
2 &Action=DescribeRegions
3 &SignatureMethod=HmacSHA256
4 &SignatureVersion=2
5 &Timestamp=2010-10-29T11%3A55%3A28
6 &Version=2009-04-04
7 &Signature=Hnb/IQ05zPmcZKCYaF34Kakn1RLVGCRH0j1/si563cg%3D
8 HTTP/1.1
9 Host: us-east-1.ec2.amazonaws.com
10 Accept-Encoding: identity
11 User-Agent: Boto/1.8d (linux2)
```

# List of Availability Zones: Response from EC2

```
1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 Content-Type: text/xml; charset=UTF-8
4 Transfer-Encoding: chunked
5 Date: Fri, 29 Oct 2010 11:55:28 GMT
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <DescribeRegionsResponse xmlns="http://ec2.amazonaws.com/doc/2009-04-04/">
9 <requestId>eaeb7fcd-0bb8-4263-b806-40170cbc9f63</requestId>
10 <regionInfo>
11 <item>
12 <regionName>eu-west-1</regionName>
13 <regionEndpoint>ec2.eu-west-1.amazonaws.com</regionEndpoint>
14 </item>
15 <item>
16 <regionName>us-east-1</regionName>
17 <regionEndpoint>ec2.us-east-1.amazonaws.com</regionEndpoint>
18 </item>
19 <item>
20 <regionName>us-west-1</regionName>
21 <regionEndpoint>ec2.us-west-1.amazonaws.com</regionEndpoint>
22 </item>
23 <item>
24 <regionName>ap-southeast-1</regionName>
25 <regionEndpoint>ec2.ap-southeast-1.amazonaws.com</regionEndpoint>
26 </item>
27 </regionInfo>
28 </DescribeRegionsResponse>
```

# End



**HOW TO GET A SOA**