8th Slide Set Cloud Computing

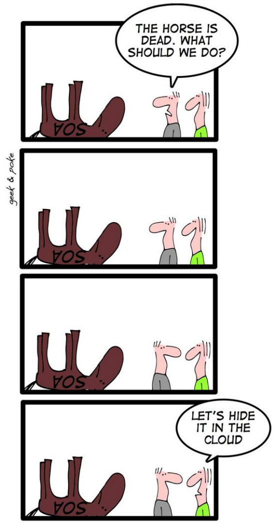
Prof. Dr. Christian Baun

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

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)


THE HORSE IS DEAD. WHAT SHOULD WE DO?
SOA
geek & poke
LET'S HIDE IT IN THE CLOUD

- 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
- The 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

Potential Applications for Web Services

Definition of the Web Services Architecture Working Group

A Web service is a software system, identified by a Uniform Resource Identifier (URI), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols

Source: `http://www.w3.org/TR/wsa-reqs`

- The most popular fields of applications for web services are. . .
  - Remote Procedure Calls
  - SOAP web services with WSDL as communication interface
  - RESTful web services

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

```perl
#!/usr/bin/perl
use strict;
use Frontier::Daemon;

sub sumAndDifference {
    my ($x, $y) = @_;
    return {'sum' => $x + $y, 'difference' => $x - $y};
}

# Call me as http://localhost:8080/RPC2
my $methods = {'sample.sumAndDifference' => \&sumAndDifference};
Frontier::Daemon->new(LocalPort => 8080, methods => $methods)
    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)

```
#!/usr/bin/perl
use strict;
use Frontier::Client;

unless (scalar(@ARGV) == 2) {
    die "usage: $0 x y\n";
}

# Make an object to represent the XML-RPC server.
my $server_url = 'http://localhost:8080/RPC2';
my $server = Frontier::Client->new(url => $server_url);

# Call the remote server and get our result.
my $result = $server->call('sample.sumAndDifference', @ARGV);
my $sum = $result->{'sum'};
my $difference = $result->{'difference'};

print "Sum: $sum, Difference: $difference\n";
```

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

XML-RPC Example (Request via HTTP POST)

```
POST /RPC2 HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost:8080
User-Agent: libwww-perl/5.808
Content-Type: text/xml
Content-Length: 199

<?xml version="1.0"?>
<methodCall>
  <methodName>sample.sumAndDifference</methodName>
  <params>
    <param>
      <value><int>2</int></value>
    </param>
    <param>
      <value><int>3</int></value>
    </param>
  </params>
</methodCall>
```

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

XML-RPC Example (Reply)

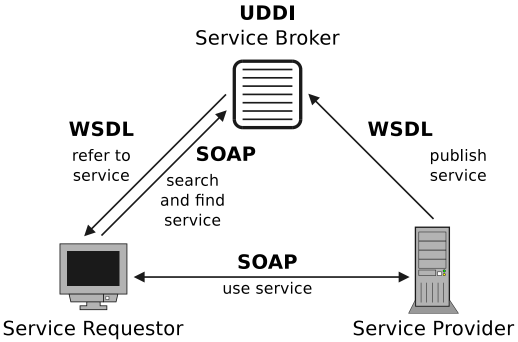
```
HTTP/1.1 200 OK
Date: Tue, 27 Jul 2010 18:19:10 GMT
Server: libwww-perl-daemon/1.39
Content-Length: 252
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>difference</name>
            <value><int>-1</int></value>
          </member>
          <member>
            <name>sum</name>
            <value><int>5</int></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

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

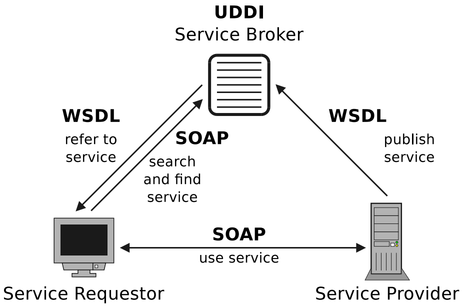
Big Web Services with SOAP in Theory

- 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

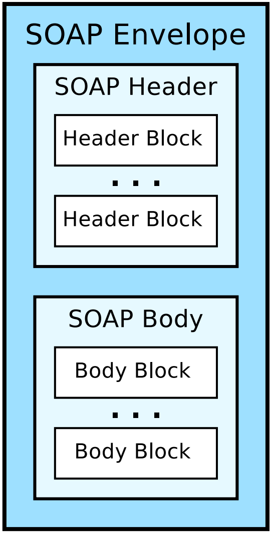


Web Services: SOAP

- 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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="http://localhost/axis/Calc.jws"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://localhost/axis/Calc.jws"
   xmlns:intf="http://localhost/axis/Calc.jws"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- portType -->
   <!-- message -->
   <!-- binding -->
   <!-- service -->
</wsdl:definitions>
```

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

WSDL Example – Part 2/5

```
<!-- definitions -->
<wsdl:portType name="Calc">
   <wsdl:operation name="add" parameterOrder="a b">
      <wsdl:input message="impl:addRequest" name="addRequest"/>
      <wsdl:output message="impl:addResponse" name="addResponse"/>
   </wsdl:operation>
</wsdl:portType>
<!-- message -->
<!-- binding -->
<!-- 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

```
<!-- definitions -->
<!-- portType -->
<wsdl:message name="addRequest">
   <wsdl:part name="a" type="xsd:int"/>
   <wsdl:part name="b" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="addResponse">
   <wsdl:part name="addReturn" type="xsd:int"/>
</wsdl:message>
<!-- binding -->
<!-- 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

```
<!-- definitions -->
<!-- portType -->
<!-- message -->
<wsdl:binding name="CalcSoapBinding" type="impl:Calc">
   <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="add">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="addRequest">
        <wsdlsoap:body
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="addResponse">
         <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost/axis/Calc.jws" use="encoded"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<!-- 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

```
<!-- definitions -->
<!-- portType -->
<!-- message -->
<!-- binding -->
<wsdl:service name="CalcService">
   <wsdl:port binding="impl:CalcSoapBinding" name="Calc">
      <wsdlsoap:address location="http://localhost/axis/Calc.jws"/>
   </wsdl:port>
</wsdl:service>
```

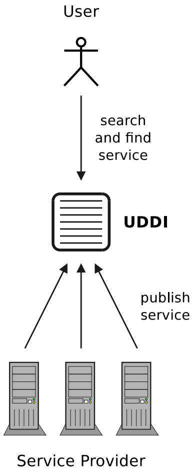
- Here, the definition of the port(s) to access the web service takes place
- Multiple access points can be specified per web service
- The web service runs locally and is located in `/axis/Calc.jws`

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 ($\Longrightarrow$ Yellow Pages)
     - Categorizations of the services, according to their purpose
  3. Green Pages
     - Contains technical information about services
     - Includes the interface descriptions

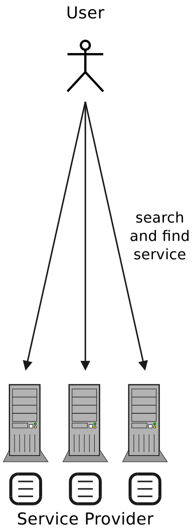
Architecture and Data Model of UDDI

- 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

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

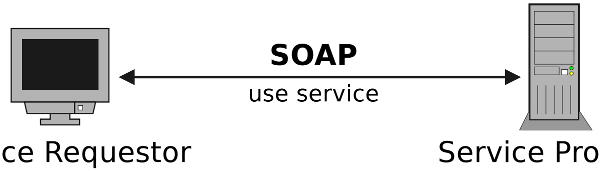
Source: Service-orientierte Architekturen mit Web Services. Elsevier (2005)

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

HTTP	CRUD Actions	SQL	Description
`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

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

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

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

```
HTTP/1.1 200 OK
x-amz-id-2: JaYLRR6nqyUpOM+eWegCmaBNC2NZJZ...etc...
x-amz-request-id: 1FCDF412571D35AA
Date: Fri, 29 Oct 2010 11:05:25 GMT
Content-Type: application/xml
Transfer-Encoding: chunked
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>af0af9137ff66f0aeeeb68dd1e50a91...usw...</ID>
    <DisplayName>christianbaun</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>christianbaun</Name>
      <CreationDate>2010-08-10T14:20:10.000Z</CreationDate>
    </Bucket>
    <Bucket>
      <Name>cloud-vorlesung-bucket</Name>
      <CreationDate>2010-10-28T21:52:46.000Z</CreationDate>
    </Bucket>
  </Buckets>
</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

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

List of Keys inside a Bucket: Response from S3

```
HTTP/1.1 200 OK
x-amz-id-2: K3xUpOC3VkCgOK2Ix9wgDRPLCysFw...usw...
x-amz-request-id: D25FD87E92C49520
Date: Fri, 29 Oct 2010 11:42:10 GMT
Content-Type: application/xml
Transfer-Encoding: chunked
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>cloud-vorlesung-bucket</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>Testdatei.txt</Key>
    <LastModified>2010-10-28T21:54:11.000Z</LastModified>
    <ETag>&quot;20437f481bb3f606f6c4bb64e997da7d&quot;</ETag>
    <Size>44</Size>
    <Owner>
      <ID>af0af9137ff66f0aeeeb68dd1e50a91...usw...</ID>
      <DisplayName>christianbaun</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>

...

</ListBucketResult>
```

Request the List of Availability Zones in EC2 via REST

- Request from the client

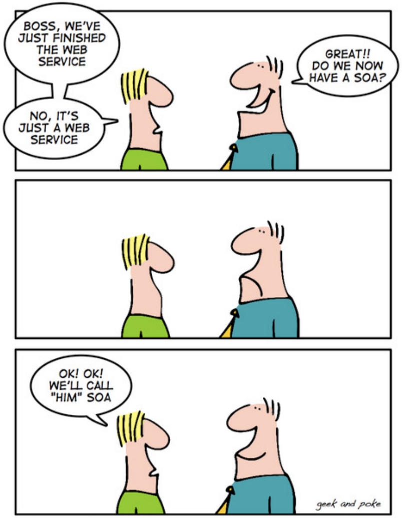
```
GET /?AWSAccessKeyId=AKIAI4QECP523R4SGJJQ
&Action=DescribeRegions
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&Timestamp=2010-10-29T11%3A55%3A28
&Version=2009-04-04
&Signature=Hnb/IQ05zPmcZKCYaF34Kaknl RLVGCRH0j1/si563cg%3D
HTTP/1.1
Host: us-east-1.ec2.amazonaws.com
Accept-Encoding: identity
User-Agent: Boto/1.8d (linux2)
```

List of Availability Zones: Response from EC2

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 29 Oct 2010 11:55:28 GMT

<?xml version="1.0" encoding="UTF-8"?>
<DescribeRegionsResponse xmlns="http://ec2.amazonaws.com/doc/2009-04-04/">
<requestId>eaeb7fcd-0bb8-4263-b806-40170cbc9f63</requestId>
  <regionInfo>
    <item>
      <regionName>eu-west-1</regionName>
      <regionEndpoint>ec2.eu-west-1.amazonaws.com</regionEndpoint>
    </item>
    <item>
      <regionName>us-east-1</regionName>
      <regionEndpoint>ec2.us-east-1.amazonaws.com</regionEndpoint>
    </item>
    <item>
      <regionName>us-west-1</regionName>
      <regionEndpoint>ec2.us-west-1.amazonaws.com</regionEndpoint>
    </item>
    <item>
      <regionName>ap-southeast-1</regionName>
      <regionEndpoint>ec2.ap-southeast-1.amazonaws.com</regionEndpoint>
    </item>
  </regionInfo>
</DescribeRegionsResponse>
```

End


BOSS, WE'VE JUST FINISHED THE WEB SERVICE
GREAT!! DO WE NOW HAVE A SOA?
NO, IT'S JUST A WEB SERVICE
OK! OK! WE'LL CALL "HIM" SOA
geek and poke

HOW TO GET A SOA