International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 6, June - 2013

Cloud Application Programming Interface Based On REST Framework.

Vijay. G. R *, Dr. A. Rama Mohan Reddy**
*(PhD Scholar Department of Computer Science & Engg, JNTUA, Anantapur, A.P, India)
** (Professor, Dept. of CSE, SVU College of Engineering, Tirupati, A.P, India)

Abstract

A Cloud Application Programming Interface (Cloud API) is a type of API that enables the development of applications and services used for the provisioning of cloud hardware, software, and platforms.
A cloud API serves as a gateway or interface that provides direct and indirect cloud services to users. In this paper we discussed about A cloud API is the core component behind any public cloud solution and is generally based primarily on the REST (Representational State Transfer)and SOAP (Simple Object Access Protocol) frameworks, as well as cross-platform and vendor specific APIs.
A cloud API interacts with a cloud infrastructure to allocate computing, storage, and network resources for requested cloud applications or services. Cloud APIs vary according to the provided service or solution, as follows: Infrastructure as a Service (IaaS): Infrastructure APIs provision raw computing and storage. Software as a Service (SaaS): Software or application APIs provision connectivity and interaction with a software suite.

1. Introduction

We live in an application economy. Increasingly, the primary digital mode of engagement between businesses and their customers, partners, and even employees is through apps. To deliver services via apps, most businesses develop Application Programming Interfaces (APIs) that support machine-to-machine interactions over the Web.
Programmable Web has determined that there are more than 5,000 APIs available on the Web today.
Cloud solutions are primarily integrated through APIs. APIs as the glue of Software as a Service (SaaS) and data gets into and out of the cloud, and from enterprise to enterprise.
It's only now that enterprises are learning best practices for running and securing APIs. It's not enough to offer an API, it needs to be reliable, scalable, and secure. Many enterprises don't really know how to offer APIs with the same security and service level as their enterprise applications.
The vast majority of developers today are trained and working in a lightweight, simple architectural style called REST (Representational State Transfer), which is ideal for the biggest sector of the app market, mobile devices.
But many of the first-generation APIs were written using SOAP (Simple Object Access Protocol) or other variations on web services. Enterprises offering APIs will need to speak REST to the outside world in order to continue to grow.

2. Cloud Overview

A few years ago, Google capitalized on the idea of using huge clusters of cheap computers and building software with a high fault tolerance, so that when its machines would go down, the overall system would still function properly. Amazon used the same approach, with a twist. The company's vast, distributed architecture lent itself to a standardized provisioning of computing resources. As the provisioning process became more standardized, and it became easier for Amazon to roll out new services, the company started selling on-demand access to its massive, distributed computing infrastructure.
For example, Go Grid, offers flexible server specifications and interconnection options. An organization provides an application infrastructure with database, security, workflow, and other capabilities already in place. Ning emphasizes simplicity, giving its users the ability to roll their own social networks without any programming knowledge.

3. Application Programming Interfaces

An Application-Programming Interface is one of the key technical facilitators of cloud computing. Besides the obvious usage of an API to access services like Simple Storage Service (S3), Third-Party Applications that add graphical user interfaces to cloud services such as Amazon Elastic Compute Cloud (EC2) and Twitter also take advantage of APIs to access the services.it's probably true, at least in the current state of how we interface with cloud

providers. Cloud computing is not simply another type of Application Service Provider (ASP) offering. The ASP and hosting providers of years gone by offered a set of software services. These services were tied to specific hardware and software configurations.

In brief, cloud computing draws its strength from its connections to the outside world, through APIs. These APIs fall into three general categories:

a. Control APIs, which allow cloud infrastructure to be added, reconfigured, or removed in real time, either by human control or programmatically based on traffic, outages, or other factors

b. Data APIs, which are the conduits through which data flow in and out of the cloud.

c. Application Functionality APIs, which enable the functionality that end users interact with, such as shopping carts, wikis, and widgets.

4. REST Resources and Architecture

Representational **S**tate **T**ransfer (REST) relies on a stateless, client-server, cacheable communications protocol and in virtually all cases, the HTTP protocol is used.

For example, you can:

• Retrieve summary information about the API versions available to you.

• Obtain detailed information about a object such as an Account or a Custom object.

• Obtain detailed information about objects, such as User or a custom object.

• Perform a query or search.

• Update or delete records.

Generally the simple REST services systems are divided as following types.

- **Resources** - The resources of your system which implement the HTTP methods POST, GET, PUT, DELETE. For each request received, it uses the modules of the utils package and the communication provided with dao, if necessary. The resources also have the task of interpret the result , possible failures and response to the client.
- **Utils** - The utility classes as also related to Data Transformation.
- **DAO** - The classes with the pattern DAO (Data Access Object), responsible for the database transactions

REST is a new architecture for web services that is having a significant impact on the industry. Most of the new public web services from large vendors (Google, Yahoo, Amazon, and Microsoft) rely on REST as the technology for sharing and merging information from multiple sources. The motivation behind REST web services is what drives the progress of technology: to make complex things simpler.

The first generation web services relied on exchanging XML packets conforming to SOAP (Simple Object Access Protocol) specification using HTTP protocol. In fact web services specification does not say that SOAP messages have to be exchanged over HTTP.This decouples SOAP messaging from the underlying communication protocol allowing for TCP/IP and other ways of exchanging SOAP

Supporters of REST architecture consider SOAP and XML to be too heavy. HTTP itself has enough capabilities for applications to communicate over the network. In reality, HTTP is what powers the Web and it has a very rich vocabulary in terms of verbs, URIs, request and response headers and Internet media types. It is also common in REST services to use JSON (JavaScript Object Notation) as a convenient, alternative to XML.

JSON being a subset of JavaScript, is an ideal data format for building pure HTML clients running in web browsers with embedded JavaScript logic and accessing REST resources in an asynchronous fashion using AJAX for highly responsive user interfaces. On the web, data is moving faster than we can browse it. Hence, there is a strong demand for programs that can find, track and monitor information coming from diverse sources including sales data, financial information, online communities, marketing campaign etc.

REST can also change the way how complex systems are architected. Traditional Service Oriented Architecture (SOA) is slowly moving towards Web Oriented Architecture (WOA) where applications gives a rich web of REST resources. Instead of a few point SOA services, enterprise data will be exposed through millions of granular REST resources, like the web itself

4.1 Support for JSON and XML

JSON (JavaScript Object Notation) is the default. You can use the HTTP ACCEPT header to select either JSON or XML or append .json or.xml to the URI (example,/Account/001D000000INjVe.json)

The JavaScript Object Notation (JSON) format is supported with UTF-8. Date-time information is in ISO8601 format. XML serialization is similar to SOAP API. XML requests are supported in UTF-8 and UTF-16, and XML responses are provided in UTF-8. Using REST, the developer must know well the four main methods: POST, GET, PUT and

DELETE and a little about the HEAD and OPTIONS.

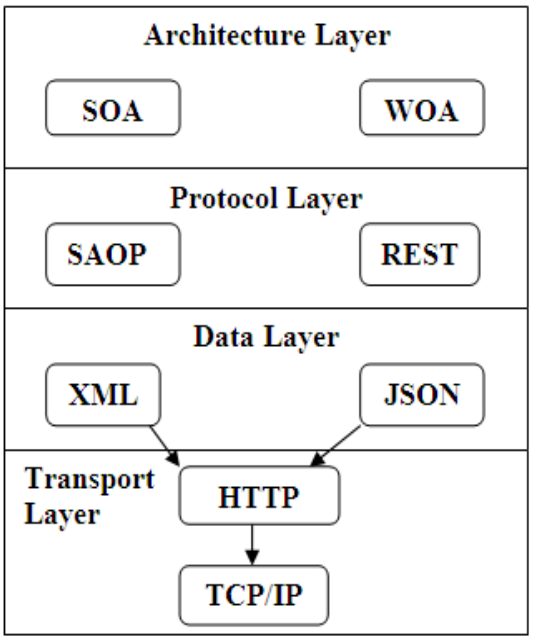

Fig.1 REST Architecture for Web Services.

POST
All the methods of adding details to the server; all the different interfaces and formats the server supports for adding data to its database.
The data are included in the body of the request. For instance, making a request POST *http://www.example.com/resources/user/*
GET
All the methods of getting data from the server; the formats and interfaces the server supports for accessing the client details.
In practice, making a request GET in *http://www.example.com/resources/user/10* will return as response the user whose the id is 10.
In the other hand if our request was to the URL *http://www.example.com/resources/user/*,we would have as response the list of all users.
PUT
All the methods for updating the data at the server; different types of interfaces and formats the server supports for adding data to the database.
The data must be sent in the body of the request. Besides, if the URI of the request doesn't not point out to a existing resource, it is allowed to create a new resource with this URI. In our example, if we wanted to update the password of the user with the login "abcdefghij", we just need to make a PUT *http://www.example.com/resources/user/10*.

DELETE
All the methods for deleting the data at the server; different types of interfaces and formats the server supports for deleting data in the database. Given the ease of design and flexibility in coding provided by REST, it has gradually become popular. Yahoo and eBay were the first ones to use REST for designing their Web services. They were later joined by popular firms such as Amazon and Google.
It deletes a specified resource and doesn't have the body. If we request the DELETE method to URI *http://www.example.com/resources/user/10* it would mean to the serve that we want it to delete the user with the id is equal to 10.
HEAD
Similar to the method GET, but without the body of the response. This method can be used to obtain the metadata of a entity target of the request, without transfer all the data to the client.
OPTIONS
Return the HTTP methods that the server supports for a specified URI. It can be used to check the features available of a web service. Making a request OPTIONS
to *http://www.example.com/resources/user/* , we would receive the attribute 'Allow' in the headers with the fields OPTIONS and POST.
However making the request OPTIONS to URI *http://www.example.com/resources/user/*, we would receive the response OPTIONS, GET, PUT, DELETE and HEAD.
When you put the wildcat (*) it is expected some response, but our method POST, using the good practices, is not mapped to accept requests with the URI finishing in 'user/*'. In other words, it doesn't make sense to request a POST to 'user/10' , since the id of the resource must be created by the server.

4.2 Response Time and All Four Functions

First, the client sent customer details in XML format to the server. The server processed the XML, connected to the database, and stored the information (POST).
Second, the client issued a GET request for customer details (GET). The GET request was followed by an update. The client sent an update request to the server for the data that had been added, using the POST function. Later, the client sent a final request to the server to delete the data that had been added and updated before.
Following is the skeleton of the code for measuring response time of the four functions in sequence:

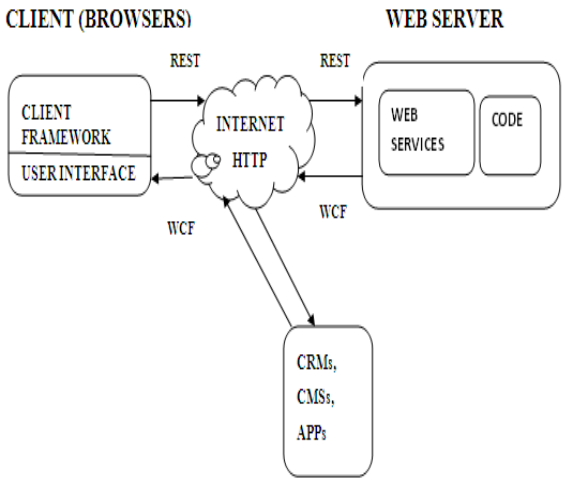

Fig.2 REST communication between Client and Server.

Client Implementation Thread class
{
Thread run method
Public void run ( )
{
POSTmethod (with correlation ID);
GETmethod (with correlation ID);
PUTmethod (with correlation ID);
DELETE method (with correlation ID);
}
POSTmethod ()
{
System time in milliseconds A, of a particular thread X; (Timer started for the
thread based on correlation ID)
Code for POST Operation with customer ID;
}
GETmethod ()
{
Code for Get Operation with customer ID;
}
PUTmethod ()
{
Code for Put Operation with customer ID;
}
DELETEmethod ()
{
Code for Delete Operation with customer ID;
System time in milliseconds A, of a particular thread X; (Timer end for the thread
based on correlation ID)
}
}

The skeleton code provides insight into the functionality needed for testing and measuring response time.

All the functions were coded with correlation IDs and customer IDs. Every time a thread initialized, it took an ID based on the correlation ID. Later, the customer ID was based on the thread ID. First, a thread was initialized and took an ID. It started executing the POST function and sent customer details to be added to the server (customer ID was based on the thread ID).
Later, the thread executed the GET function and retrieved the customer's details with the customer ID. This was followed by update (POST) and DELETE functions, based on the customer ID. The cycle involved adding customer data, getting the data added, and deleting the data added.
The starting and ending times of a thread were measured in the GET and DELETE functions (first line of code in the GET function and last line of code in the DELETE function). The difference between the starting and ending times was the execution time of the thread. Response times were taken with threads ranging from 1 to 40 (each thread had customer data), for both REST and SOAP.

5. Analysis

5.1 REST vs. SOAP Comparison Graphs

Figures 3 and 4 shows the comparison for REST vs. SOAP Response times for all four functions in a wired and then a wireless environment.

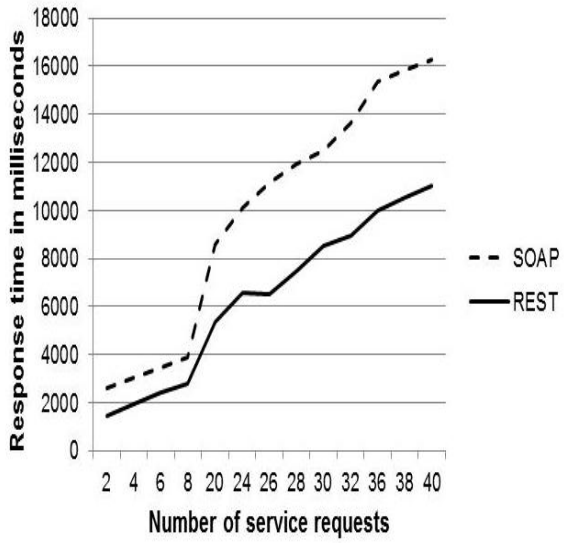

Fig.3: All Four Functions Wired SOAP vs. REST
In Figures 1 and 2, REST appears to have performed comparatively better than SOAP, for all four

functions. Among the four different functions, the GET function constitutes the majority of the response time, affecting the overall functionality and response time, accounting for the major performance difference.

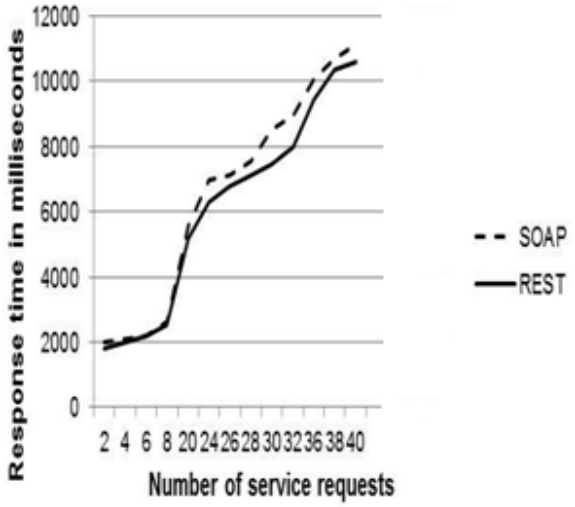

Fig.4: All Four Functions Wireless SOAP vs. REST

6. Conclusion

REST is quickly becoming the preferred technology for building arbitrary applications that communicate over the network. REST fully leverages protocols and standards that power the World Wide Web and is simpler than traditional SOAP-based web services.

With the emergence of cloud-computing and the growing interest for web hosted applications, REST-based technologies can help both in the development of rich user interface clients calling into remote servers; and in the development of actual servers for manipulating data structures in a client application written in any language or directly in the browser.

7. Reference

[1].Tekli,J.M.;Damiani,E.;Chbeir,R.;Gianini,G. SOAP Processing Performance and Enhancement" Services Computing, IEEE Transactions on 10.1109/TSC.2011.11 2012 Page(s): 387 – 403.

[2] Cesare Pautasso, Olaf Zimmermann, Frank Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision" WWW 2008, April 21–25, 2008, Beijing, China. ACM 978-1-60558-085-2/08/04.

[3].Snehal Mumbaikar, Puja Padiya, "Web Services Based On SOAP and REST Principles" International Journal of Scientific and Research Publications (IJSRP), Volume 3, Issue 5, May 2013, ISSN 2250-3153.

[4].Kishor Wagh,Dr. Ravindra Thool "A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host" Journal of Information Engineering and Applications (JIEA), Vol 2, No.5, 2012, ISSN 2225-0506.

[5].http://aws.amazon.com/security/Amazon Web Services: Overview of Security Processes, March 2013.

[6]. Davis John and Dr. Rajasree M. S. "RESTDoc: Describe, Discover and Compose RESTful Semantic Web Services using Annotated Documentations" International Journal of Web & Semantic Technology (IJWesT) Vol.4, No.1, January 2013.

[7].Crockford, "RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON)", Network Working Group, July 2006.

[8].Doug Tidwell, Cloud Computing Evangelist, "Keeping Your Options Open, Even if the Cloud is not" IBM Corporation. 2011.

[9].Brian Adler Professional Services Architect "Designing Private and Hybrid Clouds: Architectural Best Practices" 2012 Right Scale, Inc.