A **RESTful API** is an application program interface (API) that uses HTTP requests to **GET**, **PUT**, **POST** and **DELETE** data.

A RESTful API -- also referred to as a **RESTful web service** -- is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST leverages less bandwidth, making it more suitable for internet usage. An API for a website is code that allows two software programs to communicate with each another. The API spells out the proper way for a developer to write a program requesting services from an operating system or other application.

The REST used by browsers can be thought of as the language of the internet. With cloud use on the rise, APIs are emerging to expose web services. REST is a logical choice for building APIs that allow users to connect and interact with cloud services. RESTful APIs are used by such sites as Amazon, Google, LinkedIn and Twitter.

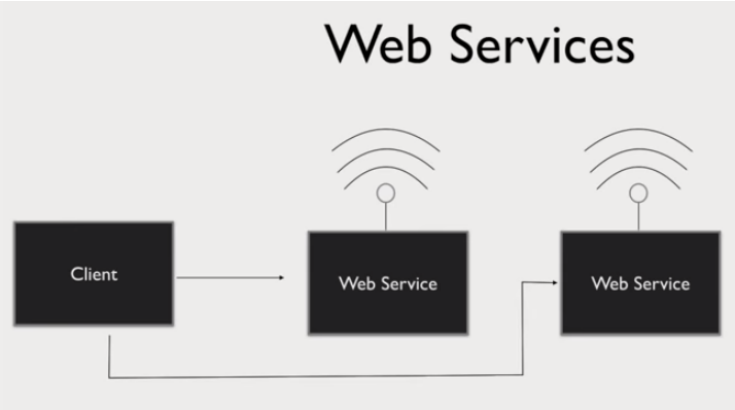
How RESTful APIs work

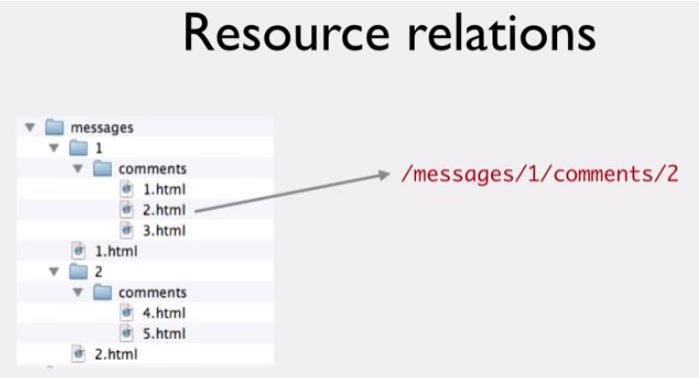
A RESTful API breaks down a transaction to create a series of small modules. Each module addresses a particular underlying part of the transaction. This modularity provides developers with a lot of flexibility, but it can be challenging for developers to design from scratch. Currently, the models provided by Amazon Simple Storage Service, Cloud Data Management Interface and OpenStack Swift are the most popular.

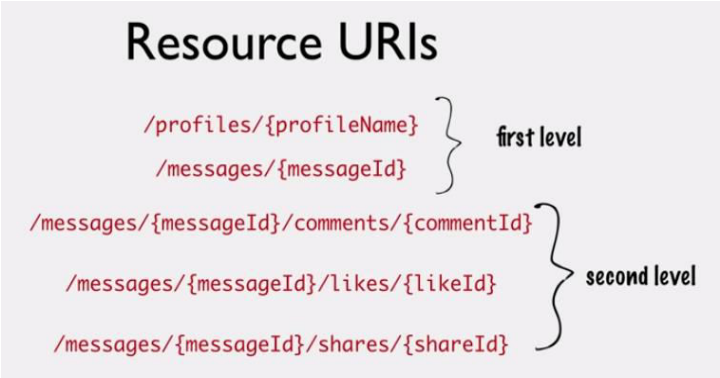
A RESTful API explicitly takes advantage of HTTP methodologies defined by the RFC 2616 protocol. They use GET to retrieve a resource; PUT to change the state of or update a resource, which can be an object, file or block; POST to create that resource; and DELETE to remove it.

With REST, networked components are a resource you request access to -- a black box whose implementation details are unclear. The presumption is that all calls are stateless; nothing can be retained by the RESTful service between executions.

Because the calls are stateless, REST is useful in cloud applications. Stateless components can be freely redeployed if something fails, and they can scale to accommodate loadchanges. This is because any request can be directed to any instance of a component; there can be nothing saved that has to be remembered by the next transaction. That makes REST preferred for web use, but the RESTful model is also helpful in cloud services because binding to a service through an API is a matter of controlling how the URL is decoded. Cloud computing and microservices are almost certain to make RESTful API design the rule in the future.







Note, **What is a Maven artifact?**

In Maven terminology, the **artifact** is the resulting output of the **maven build**, generally a `jar` , `war` or other executable file.
In general software terms, an "artifact" is something produced by the software development process, whether it be software related documentation or an executable file.

Artifacts in maven are identified by a coordinate system of *groupId, artifactId*, and *version*. Maven uses the `groupId`, `artifactId`, and `version` to identify dependencies (usually other jar files) needed to build and run your code.

Note, What is Archetype?

In short, Archetype is a Maven project **templating toolkit**. An archetype is defined as *an original pattern or model from which all other things of the same kind are made*.

The name fits as we are trying to provide a system *that provides a consistent means of generating Maven projects.* Archetype will help authors create Maven project templates for users and provides users with the means to generate parameterized versions of those project templates.

Using archetypes provides a great way to enable developers quickly in a way consistent with best practices employed by your project or organization. Within the Maven project, we use archetypes to try and get our users up and running as quickly as possible by providing a sample project that demonstrates many of the features of Maven, while introducing new users to the best practices employed by Maven. In a matter of seconds, a new user can have a working Maven project to use as a jumping board for investigating more of the features in Maven. We have also tried to make the Archetype mechanism additive, and by that we mean allowing portions of a project to be captured in an archetype so that pieces or aspects of a project can be added to existing projects. A good example of this is the Maven site archetype. If, for example, you have used the quick start archetype to generate a working project, you can then quickly create a site for that project by using the site archetype within that existing project. You can do anything like this with archetypes.

You may want to standardize J2EE development within your organization, so you may want to provide archetypes for EJBs, or WARs, or for your web services. Once these archetypes are created and deployed in your organization's repository, they are available for use by all developers within your organization.