Infrastructure Architecture: Whitelisting with JSON and API Keys

Lori MacVittie, 2011-12-10

Application delivery infrastructure can be a valuable partner in architecting solutions ….

AJAX and JSON have changed the way in which we architect applications, especially with respect to their ascendancy to rule the realm of integration, i.e. the API. Policies are generally focused on the URI, which has effectively become the exposed interface to any given application function. It's REST-ful, it's service-oriented, and it works well.

Because we've taken to leveraging the URI as a basic building block, as the entry-point into an application, it affords the opportunity to optimize architectures and make more efficient the use of compute power available for processing. This is an increasingly important point, as capacity has become a focal point around which cost and efficiency is measured. By offloading functions to other systems when possible, we are able to increase the useful processing capacity of an given application instance and ensure a higher ratio of valuable processing to resources is achieved.

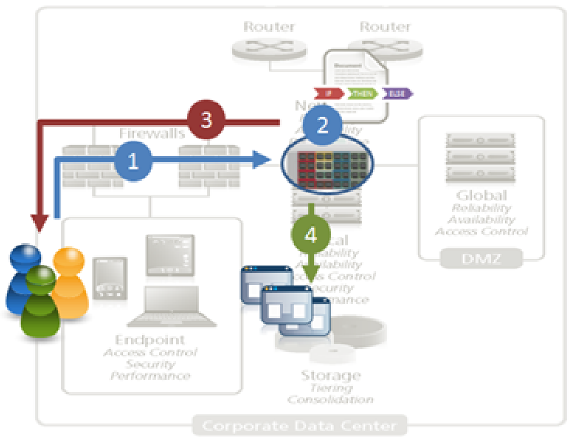
The ability of application delivery infrastructure to intercept, inspect, and manipulate the exchange of data between client and server should not be underestimated. A full-proxy based infrastructure component can provide valuable services to the application architect that can enhance the performance and reliability of applications while abstracting functionality in a way that alleviates the need to modify applications to support new initiatives.

AN EXAMPLE

Consider, for example, a business requirement specifying that only certain authorized partners (in the integration sense) are allowed to retrieve certain dynamic content via an exposed application API. There are myriad ways in which such a requirement could be implemented, including requiring authentication and subsequent tokens to authorize access – likely the most common means of providing such access management in conjunction with an API. Most of these options require several steps, however, and interaction directly with the application to examine credentials and determine authorization to requested resources. This consumes valuable compute that could otherwise be used to serve requests.

An alternative approach would be to provide authorized consumers with a more standards-based method of access that includes, in the request, the very means by which authorization can be determined. Taking a lesson from the credit card industry, for example, an algorithm can be used to determine the validity of a particular customer ID or authorization token. An API key, if you will, that is not stored in a database (and thus requires a lookup) but rather is algorithmic and therefore able to be verified as valid without needing a specific lookup at run-time. Assuming such a token or API key were embedded in the URI, the application delivery service can then extract the key, verify its authenticity using an algorithm, and subsequently allow or deny access based on the result.

This architecture is based on the premise that the application delivery service is capable of responding with the appropriate JSON in the event that the API key is determined to be invalid. Such a service must therefore be network-side scripting capable. Assuming such a platform exists, one can easily implement this architecture and enjoy the improved capacity and resulting performance boost from the offload of authorization and access management functions to the infrastructure.



1. A request is received by the application delivery service.

2. The application delivery service extracts the API key from the URI and determines validity.

3. If the API key is not legitimate, a JSON-encoded response is returned.

4. If the API key is valid, the request is passed on to the appropriate web/application server for processing.

Such an approach can also be used to enable or disable functionality within an application, including live-streams. Assume a site that serves up streaming content, but only to authorized (registered) users. When requests for that content arrive, the application delivery service can dynamically determine, using an embedded key or some portion of the URI, whether to serve up the content or not. If it deems the request invalid, it can return a JSON response that effectively "turns off" the streaming content, thereby eliminating the ability of non-registered (or non-paying) customers to access live content.

Such an approach could also be useful in the event of a service failure; if content is not available, the application delivery service can easily turn off and/or respond to the request, providing feedback to the user that is valuable in reducing their frustration with AJAX-enabled sites that too often simply "stop working" without any kind of feedback or message to the end user.

The application delivery service could, of course, perform other actions based on the in/validity of the request, such as directing the request be fulfilled by a service generating older or non-dynamic streaming content, using its ability to perform application level routing.

The possibilities are quite extensive and implementation depends entirely on goals and requirements to be met.

Such features become more appealing when they are, through their capabilities, able to intelligently make use of resources in various locations. Cloud-hosted services may be more or less desirable for use in an application, and thus leveraging application delivery services to either enable or reduce the traffic sent to such services may be financially and operationally beneficial.

ARCHITECTURE is KEY

The core principle to remember here is that ultimately infrastructure architecture plays (or can and should play) a vital role in designing and deploying applications today. With the increasing interest and use of cloud computing and APIs, it is rapidly becoming necessary to leverage resources and services external to the application as a means to rapidly deploy new functionality and support for new features. The abstraction offered by application delivery services provides an effective, cross-site and cross-application means of enabling what were once application-only services within the infrastructure. This abstraction and service-oriented approach reduces the burden on the application as well as its developers.

The application delivery service is almost always the first service in the oft-times lengthy chain of services required to respond to a client's request. Leveraging its capabilities to inspect and manipulate as well as route and respond to those requests allows architects to formulate new strategies and ways to provide their own services, as well as leveraging existing and integrated resources for maximum efficiency, with minimal effort.

Related blogs & articles:

- HTML5 Going Like Gangbusters But Will Anyone Notice?
- Web 2.0 Killed the Middleware Star
- The Inevitable Eventual Consistency of Cloud Computing

- Let's Face It: PaaS is Just SOA for Platforms Without the Baggage
- Cloud-Tiered Architectural Models are Bad Except When They Aren't
- The Database Tier is Not Elastic
- The New Distribution of The 3-Tiered Architecture Changes Everything
- Sessions, Sessions Everywhere