Programmability in the Network: Versioning Patterns

Lori MacVittie, 2013-01-07

#devops API and application versioning becomes a requirement when dev is agile.

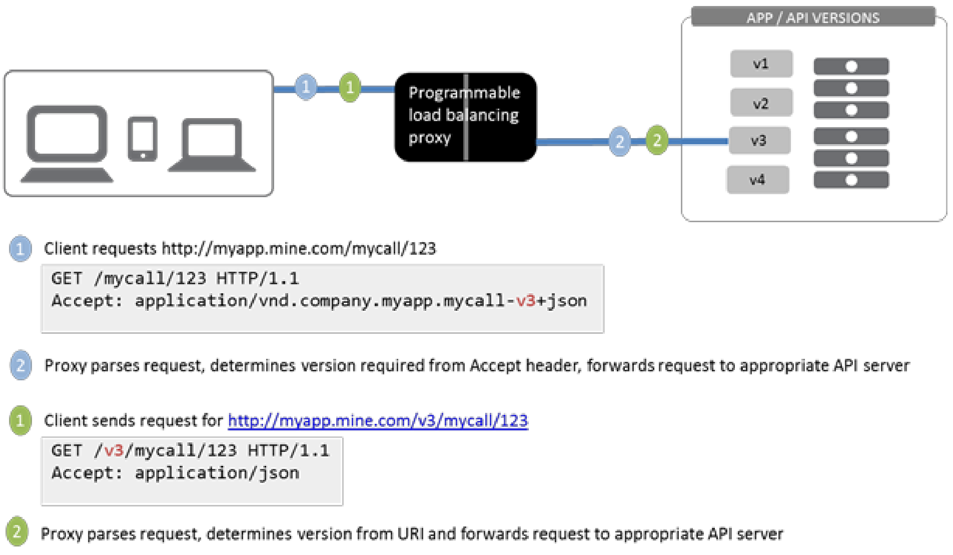
Agile development methodologies advocate a rapid release cycle. Facebook, for example, noted last year that it pushes code live twice a day. Organizations may never push code that often, but they do push new versions, new features, and new APIs on a much more frequent cycle than has been done in the past. In the old days, such rollouts were few and far between, carefully coordinated to ensure that clients were updated at the same time. It was, for the entire organization, a task of gargantuan proportions. Migration when versioning becomes critical to the functionality (or stability or security or performance) becomes problematic in an agile environment. It's more likely that you'll be supporting at least two if not three or more versions of the same API or application and encouraging migration over time.

Assuming this is the status quo today, it begs the question *how* an organization deploys and maintains multiple versions of the same application or API without also maintaining multiple versions of the entire application delivery infrastructure chain. Certainly this is an option, but an operationally (and capitally) expensive one. A much better option is to leverage programmability in the network to implement support for versioning; ensuring that clients and applications or APIs match up.

The goal of implementing a versioning pattern in infrastructure is fairly straightforward: run multiple versions in parallel to enable a transitional client migration strategy. There are two common methods of achieving this goal:

- URI-based versioning
- HTTP Header-based versioning

Both are easily supported by most programmable load balancing proxies.



1 Client requests http://myapp.mine.com/mycall/123

```
GET /mycall/123 HTTP/1.1
Accept: application/vnd.company.myapp.mycall-v3+json
```

2 Proxy parses request, determines version required from Accept header, forwards request to appropriate API server

1 Client sends request for http://myapp.mine.com/v3/mycall/123

```
GET /v3/mycall/123 HTTP/1.1
Accept: application/json
```

2 Proxy parses request, determines version from URI and forwards request to appropriate API server

There are two primary methods of versioning applications and APIs and both are equally supported by programmable load balancing proxies. In both cases, the proxy must be able to intercept and inspect the request. It acts as an intermediary; as far as the client is concerned, it *is* the ultimate endpoint. Reality is that it is only the penultimate endpoint, as the actual destination must be determined based on URI or HTTP header values (or a combination thereof).

In the network demesne we call this HTTP Message Steering. It is the ability of an intelligent intermediary to parse messages - not packets, but messages - and from that determine where to direct the request. This enables the proxy to make decisions based on a number of factors. While versioning patterns generally make use of HTTP URI or Accept header values, there's no reason this information could not be carried within the message (payload) in an XML element, e.g. <version>3</version>.

By placing the onus for parsing and choosing the version appropriate endpoint on the load balancing proxy, developers can roll out new versions without also rolling out methods of diverting requests to the right version, and the proxy can support virtually thousands of versions simultaneously*.

Using programmability in the network to manage versioning also provides a centralized, single point of entry through which versions can ultimately be deprecated and decommissioned. An intelligent, programmable load balancing proxy can also respond directly to the client, which means if a version is decommissioned the proxy can redirect the client to a new version, to a support page or to the new version of the client instead. This eliminates application black-holes and frustrated customers who may not know they need to migrate to a new version. Resources that might have been necessary to support this type of functionality without a programmable load balancing proxy can be re-provisioned to support current and new versions instead.

Programmability in the network supports agile development methodologies by providing a scalable, flexible and logical point in the network where decisions regarding version handling can be quickly made. Devops, in conjunction with developers, can quickly roll out new releases and simultaneously manage multiple versions with alacrity and thus eliminate complexity and costs in the application infrastructure.

* Obviously supporting more than a few versions of the same application or API is not recommended, but hey - if you want to, knock yourself out.