



Robert Broeckelmann · Follow

8 min read · Mar 8, 2016

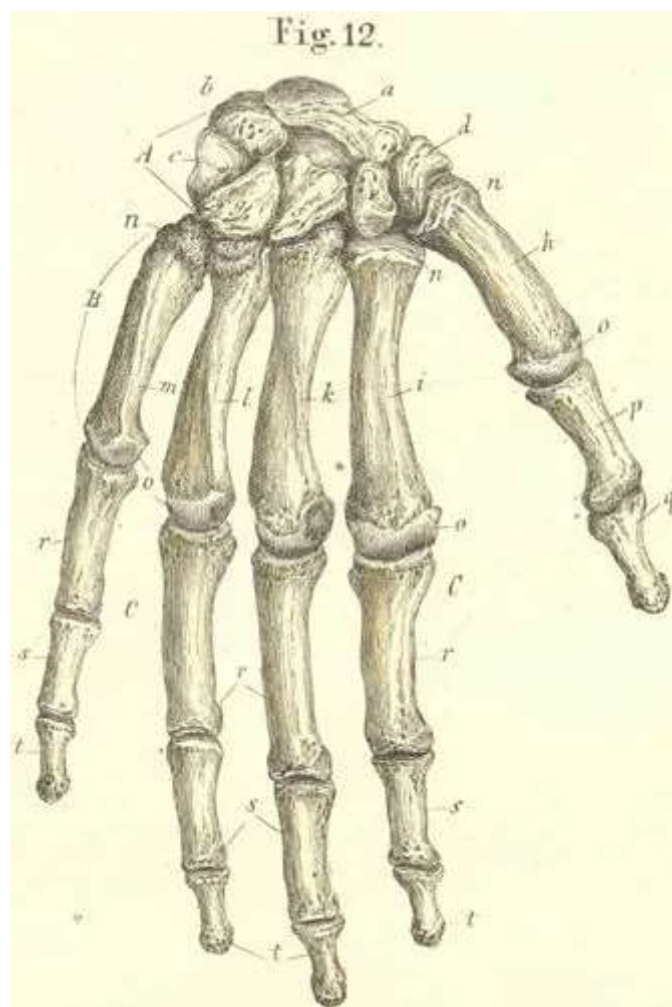


25



The Anatomy of an API Management Solution

This post was originally published as “The Anatomy of an API Management Solution” on the Levvel Blog.



In my last post, I defined what API Management is; before that, we explored what APIs are. The API Management space is crowded at the moment — there are many vendors offering solutions. To make it even more complicated, each vendor may have their own terminology. I wrote this article in order to provide enough information from which an organization can begin the evaluation process of the various API Management products. What do these products do exactly? What do they look like? From a practical standpoint, an API Management solution contains three components:

- API Gateway
- Developer Portal
- Management Portal

Every vendor has these parts although their marketing material or documentation may call them something different. For the larger, more-established companies operating in this space, these pieces were probably built from or on top of existing components (WebSphere DataPower acts as the API Gateway for IBM API Management).

API Gateway (Run-Time Gateway)

The API Gateway is the run-time component of the API Management solution. All API requests are directed through this layer of the system. It is the front door for all API traffic and the processing policies defined by the administrator are enforced here. Processing policies could include:

- security (authentication and authorization)
- JSON Threat Protection

- XML Threat Protection
- SQL Injection Threat Protection
- SLA enforcement
- flow control
- simple data transformations (probably involving changes between XML<->JSON, XML<->XML, or JSON<->JSON)
- simple protocol transformations (HTTP<->HTTPS, REST<->SOAP, maybe others)
- request validation
- schema validation
- error handling
- error reporting
- statistics logging
- many others

Developer Portal

The Developer Portal is the central point of engagement for the development community using your organization's APIs. The developers using the portal could be inside your organization, in different lines of business, from business partners, or third parties who use your APIs in their products. The Developer Portal provides self-service to:

- developer registration
- application definitions

- subscription to APIs
- viewing API documentation
- security setup
- others

The key to an effective developer experience in the Developer Portal is self-service — it is the beginning of a good story for users of your APIs. There shouldn't be a ten-business-day expected turn-around to provision a new account. If your organization functions this way, know that it does not have to be this way. Right now, you are probably thinking this sounds like DevOps; in part it is. I view an API Management solution's Developer Portal as the starting point for kicking off many of the tasks one may need to do get an application stood up (provision VMs, load balancers VIPs, IPs, firewall rules, test users, groups, authorization policy, etc, etc). But, that is a much larger picture and may very well step on someone else's toes within your organization.

After the registration process, a well-integrated Developer Portal with the organization's identity stack will allow Single Sign On (SAML-based, OAuth2-based, something else, doesn't matter that much, though modern applications now tend to use OAuth2 Implicit Grant for Web App SSO) for future access. Depending on the product, the developer may have to define an organization that serves as an umbrella object for the constructs that will be defined (e.g. IBM Websphere API Management has this concept in its Developer Portal, Apigee Edge does not). Next, the developer will have to define an application(s) — this could include an application name, application URL, and other meta-data. This will generate a client identifier (and potentially a client secret) — this corresponds to pieces needed by the

OAuth2 spec and will be important later. The mechanism that tracks the application definitions, client identifiers, client secrets, and other application meta-data will be acting as the OAuth2 Identity Provider (IdP). Most API Management Solutions come bundled with at least a basic OAuth2 IdP — probably much more than basic capabilities outlined in the OAuth 2.0 spec (e.g. IBM API Management and Apigee Edge have numerous features in this space). If a client identifier and client secret are not present (as concepts), then this means that OAuth2 isn't used and there will likely be a subscription key issued to each application definition — in the Apigee implementation of subscription keys, the key is simply the client identifier. Since there are no real rules regarding how it must be done, the basis of the API Security Model this product (or hosting platform) uses could be completely different, but typically this won't be the case. Next, the application must subscribe to an API or APIs that it wishes to invoke.

At this point, we must switch gears for a moment, to describe two other concepts in the Developer Portal: the APIs and the API collection abstraction. Anyone familiar with CA/Layer7 API Management, IBM API Management, or Apigee Edge is probably wondering what is an API collection abstraction? That is my term to collectively describe IBM API Management Plans, Apigee Edge API Products, and CA API Management API Plans. Each of these product-specific concepts provide the same basic ability to group pieces of an API or multiple APIs into a common unit that can have security, rate-limiting, and other policy applied and can then be exposed to subscribing applications. Now, the Developer Portal isn't of much use to anyone without APIs. Once an API is defined in the Management Console (more on that below) and exposed on the API Gateway (described above), it will also be exposed in the Developer Portal. From there, an administrator (could be the same admin who exposed it in the Management Console initially or a distinct role) can map the API to one of the API

collection abstractions described above. Each API will have documentation (e.g. Swagger) attached to it, server-side rate-limiting information, and other API meta-data associated with it. This documentation could be stored in the Developer Portal or it could come from a repository like WebSphere Services Registry and Repository (WSRR). If the latter, then the Developer Portal would need access to an API to pull down the meta-data for APIs; ideally, this would involve caching this information on the Developer Portal and periodically caching it. Documentation and meta-data could also be attached to the API collection abstraction concept.

Coming back to the consumer side of the Developer Portal story, the application must subscribe to APIs or the API collection abstraction I just described. Typically, the abstraction is utilized, but, the subscription could be directly to the APIs — depends on the capabilities of the product and how your organization wants to present APIs to the developer. If all of the APIs are publicly accessible, then there is no need for a detailed security policy to be applied to API collections. At this point, the developer is allowed to subscribe to any API that is desired, view the documentation, and invoke the API as required. On the other hand, if an API is not public, then the API developer (as opposed to Application Developer) or an administrator needs to define the appropriate security policy for each API or collection. As an example, maybe a collection exposes all of the read-only operations on an API or set of APIs. Maybe another collection exposes all of the update, create, and delete functionality. It could potentially become much more sophisticated. Such a security policy would require some type of approval process within the organization that owns the APIs before any application could be granted access. There are two places where access is being granted: in the developer portal to view the API or collections of APIs' documentation (and meta-data) and at run-time in the API Gateway.

This description of security assumes that access to API documentation is limited according to some set of rules based upon the roles the various developers and/or their applications play. Depending on the security model, that may not be necessary. Depending on the nature of the APIs and the use cases, that may not be desired.

Likewise, this security model assumes that the authenticated principal (the authenticated entity) that the run-time authorization decision is based upon is the application and not the end-user — this is accomplished through the client identifier, client secret, and possibly an API Key (or Subscription Key). Especially for a large organization, authorization decisions may be based upon the end-user (not the calling application). In this case, defining who can see API documentation based upon a security role assigned to the application likely has little meaning.

At this point, the developer can access the documentation for APIs their applications are subscribed to and begin making API requests against the Run-Time Gateway.

The Developer Portal will allow a developer to view analytics about their API usage, response time, status codes, and other details in the aggregate and per application (and probably other views). Most of the products on the market will also expose statistics about the developers usage of the Developer Portal. Depending on the product, the analytics screens may be customizable.

Most Developer Portal products will also have a forums feature for users to ask/answer questions, post articles, make suggestions, and interact with the API community and its creators. This can be an invaluable source of user feedback for API design and the overall developer experience. For some of the more heavily regulated industries and government organizations, this

feature may present significant compliance issues. Usage would have to be dealt with on a case-by-case basis.

Independent of the developer experience, the Developer Portal will have the ability to manage users, applications, APIs, API collections, and other constructs as needed. It will also have the ability to manage the look and feel of the Developer Portal (itself) to some extent — this will likely also involve coding depending on the vendor. For example, the Apigee Developer Portal is based on Drupal; Apigee offers numerous modules out-of-the-box to do interesting and common things. There will be other administration aspects that differ by product.

Management Portal

This brings us to the Management Portal; this component drives the activities and capabilities of the other two.

The Management Portal allows administrators to define APIs, deploy APIs, and promote APIs through the environments. It allows for the definition of environments such as dev, test, and production. How the environment concept is implemented varies between vendor products.

User and Roles can be defined for accessing the Management Portal. This can also be delegated to a federation server using Web App SSO (typically SAML or OAuth-based).

An API Developer (or possibly a distinct role of API Policy Developer), would use the Management Portal to define and apply policies (listed above) for each API.

The Management Portal will also contain some type of business analytics and system monitoring component — my experience with most of these products is that the analytics and monitoring capabilities are a good start, but may still require more (more on that later).

The Management Portal typically allows administrators to control what is exposed in the Developer Portal regarding APIs (but, most control is from the Developer Portal itself).

As always, share your thoughts and questions below.

API

Api Gateway

**Written by Robert Broeckelmann**

Follow

1.99K Followers · 1 Following

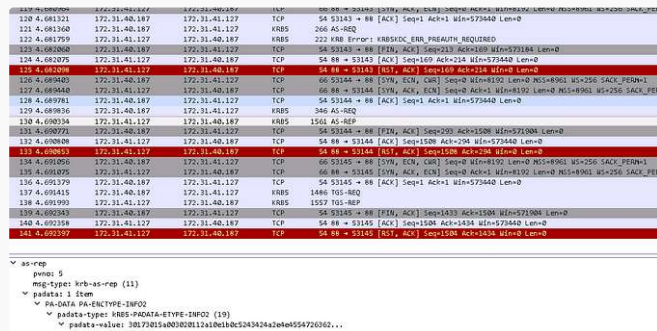
My focus within Information Technology is API Management, Integration, and Identity—especially where these three intersect.


No responses yet

What are your thoughts?

Respond

More from Robert Broeckelmann




 Robert Broeckelmann

Kerberos Wireshark Captures: A Windows Login Example

This blog post is the next in my Kerberos and Windows Security series. It describes the...


May 17, 2018  254  4  



 Robert Broeckelmann

HTTP POST vs GET: Is One More Secure For Use In REST APIs?

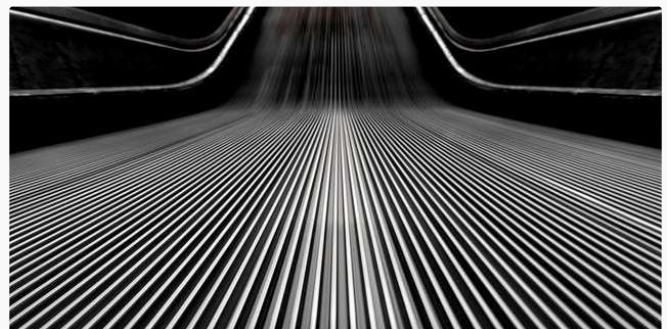



 Robert Broeckelmann

OpenID Connect Logout

The OpenID Connect (OIDC) family of specs supports logout (from a single application)...

Jul 12, 2017  490  5  



 Robert Broeckelmann

Authentication vs. Federation vs. SSO

The use of HTTP POST vs HTTP GET for read-only (or query) operations in REST APIs...

Feb 6, 2021 🖱 78



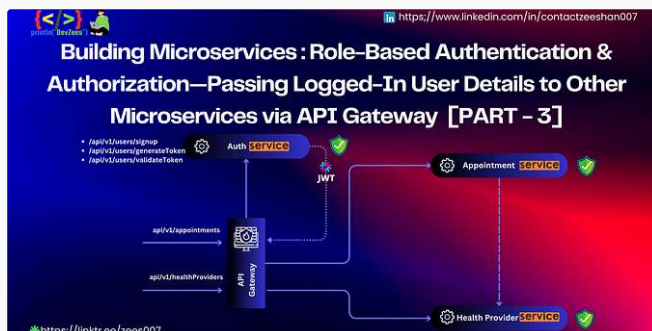
Authentication. Federation. Single Sign On (SSO). I've mentioned these concepts many...

Sep 24, 2017 🖱 859 💬 6



See all from Robert Broeckelmann

Recommended from Medium



In Level Up Coding by Zeeshan Adil

Building Microservices [PART-3]: Role-Based Authentication &...

Welcome Back to DevZees ❤️

★ Nov 12 🖱 106



In Stackademic by Crafting-Code

I Stopped Using Kubernetes. Our DevOps Team Is Happier Than Ever

Why Letting Go of Kubernetes Worked for Us

★ Nov 19 🖱 3.5K 💬 113



Lists



Coding & Development

11 stories · 926 saves



data science and AI

40 stories · 296 saves



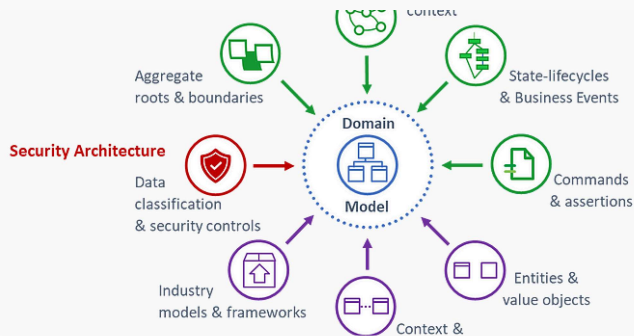
Company Offsite Reading List

8 stories · 170 saves



Natural Language Processing

1841 stories · 1463 saves



In API Central by TRGoodwill

API Design Practice

A practical guide to API QA and the design of stable, coherent and composable business...

May 9, 2023

1K

11



In Beyond Agile Leadership by Eiki Takeuchi

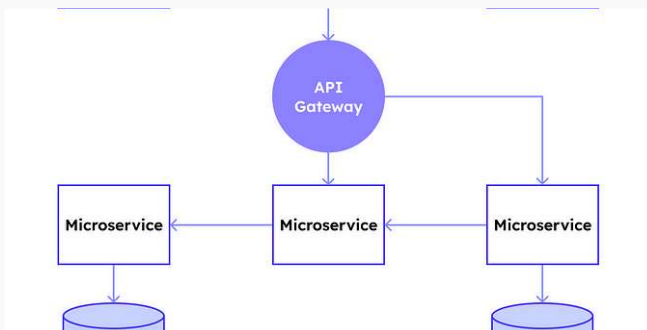
Why Hiring High-Performance Developers is the Biggest Mistake...

When I was working as a software engineer in Japan, I saw numerous smart and talented...

Nov 11

368

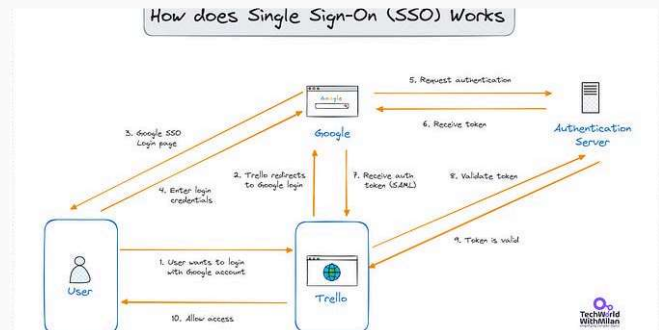
32



In Jungletronics by J3

Building a Secure Authentication Server and API Gateway for...

Step-by-Step Guide to Setting Up Token-Based Security, Scopes, and API Gateways f...




Dr Milan Milanović


How does Single Sign-On (SSO) work?

Single Sign-On (SSO) is an authentication process that allows users to access multiple...

Oct 28  24

 Mar 24  1.3K  10

See more recommendations