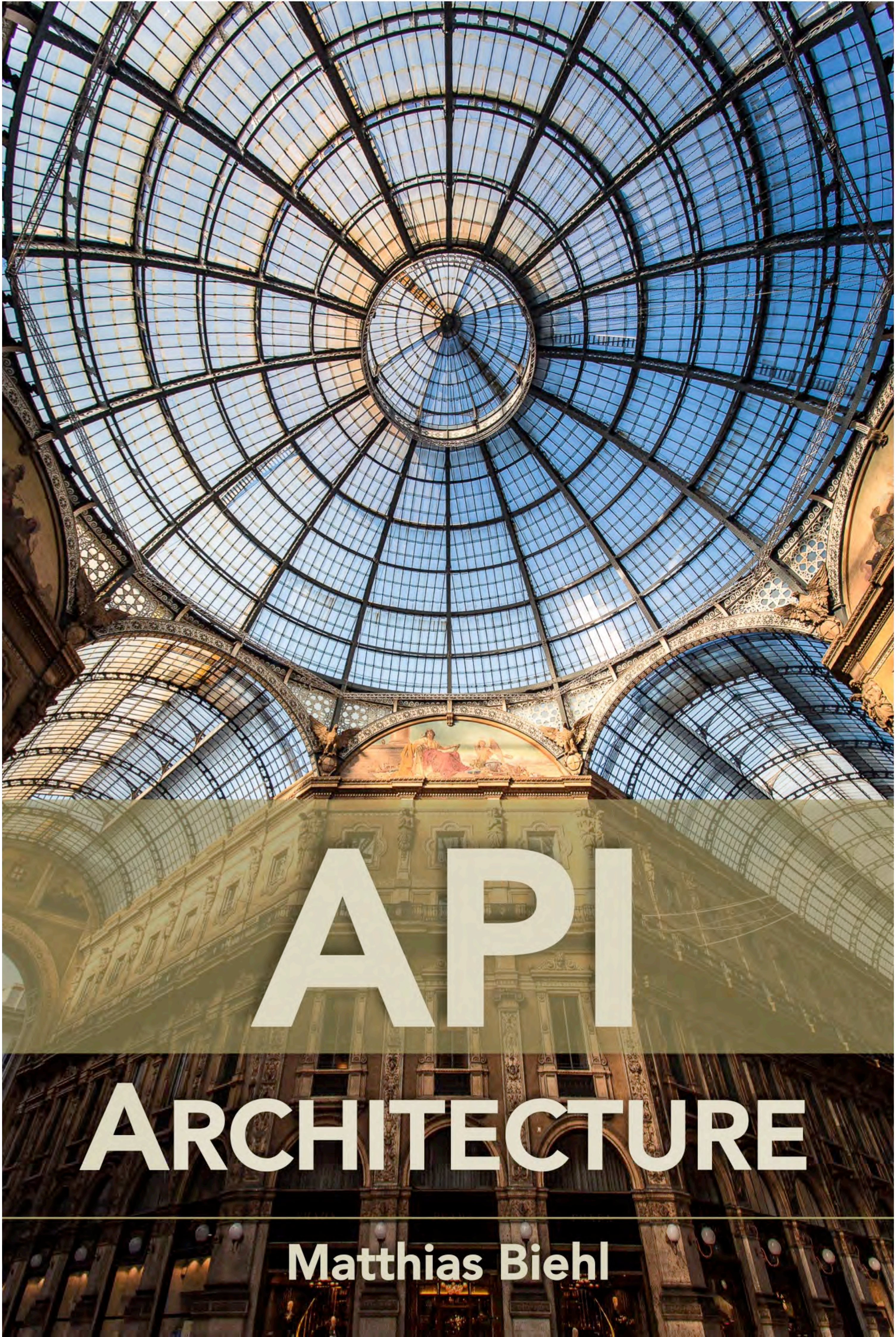

API
ARCHITECTURE

Matthias Biehl

API Architecture

The Big Picture for Building APIs

by Matthias Biehl

API University Series
www.api-university.com

API-University Press

ISBN-13: 978-1508676645
ISBN-10: 150867664X

Synopsis

Looking for the big picture of building APIs? This book is for you!

Building APIs that consumers love should certainly be the goal of any API initiative. However, it is easier said than done. It requires getting the architecture for your APIs right. This book equips you with both foundations and best practices for API architecture. This book presents best practices for putting an infrastructure in place that enables efficient development of APIs.

This book is for you if you want to understand the big picture of API design and development, you want to define an API architecture, establish a platform for APIs or simply want to build APIs your consumers love.

This book is NOT for you, if you are looking for a step-by step guide for building APIs, focusing on every detail of the correct application of REST principles. In this case I recommend the book API Design of the API-University Series.

What is API architecture? Architecture spans the bigger picture of APIs and can be seen from several perspectives:

API architecture may refer to the architecture of the complete solution consisting not only of the API itself, but also of an API client such as a mobile app and several other components. API solution architecture explains the components and their relations within the software solution.

API architecture may refer to the technical architecture of the API platform. When building, running and exposing not only one, but several APIs, it becomes clear that certain building blocks of the API, runtime functionality and management functionality for the API need to be used over and over again. An API platform provides an infrastructure for developing, running and managing APIs.

API architecture may refer to the architecture of the API portfolio. The API portfolio contains all APIs of the enterprise and needs to be managed like a product. API portfolio architecture analyzes the functionality of the API and organizes, manages and reuses the APIs.

API architecture may refer to the design decisions for a particular API proxy. To document the design decisions, API description languages are used. We explain the use of API description languages (RAML and Swagger) on many examples.

This book covers all of the above perspectives on API architecture. However, to become useful, the architecture needs to be put into practice. This is why this book covers an API methodology for design and development. An API methodology provides practical guidelines for putting API architecture into practice. It explains how to develop an API architecture into an API that consumers love.

A lot of the information on APIs is available on the web. Most of it is published by vendors of API products. I am always a bit suspicious of technical information pushed by product vendors. This book is different. In this book, a product-independent view on API architecture is presented.

The API-University Series is a modular series of books on API-related topics. Each book focuses on a particular API topic, so you can select the topics within APIs, which are relevant for you.

Keywords: API, API Management, API Architecture, Integration, API Description Languages, RAML, Swagger

Table of Contents

Introduction 15
What is an API? 15
Why APIs? 16
How are APIs used? 18
How to build APIs? 18
What is API Architecture? 20
How to put API Architecture into Practice? 21
Why is API Architecture Important? 22
API Solution Architecture 25
Types of API Solutions 26
Mobile Solutions 26
Cloud Solutions 27
Web Solutions 27
Integration Solutions 28
Multi-Channel Solutions 29
Smart TV Solutions 29
Internet of Things 30
Stakeholders in API Solutions 31
API Providers 31
API Consumers 33
End Users 33
API-related Design Decisions 34
What do all types of API solutions have in common? 34
Functionality in the Client or in the API? 35
Use an existing API or build a new API? 35
How to choose a third party API? 36
API Platform Architecture 39
Overview 40

API Development Platform 41
Library of API Building Blocks 42
Language for Implementing APIs 43
Language for Designing APIs 44
API Runtime Platform 45
API Engagement Platform 46
API Platform Configurations and Interactions 48
Environments 48
API Platform Deployment Models 49
Interactions between the Platforms 49
Surrounding Systems 51
Load Balancers and Firewalls 53
Identity and Access Management Infrastructure 53
Existing Functionality in Backends 54
New Functionality 55
Enterprise Service Buses and SOA Platforms 55

API Portfolio Architecture 59
Requirements 59
Consistency 59
Reuse 60
Customization 61
Discoverability 61
Longevity 62
Governance 62
Consistency 63
Consistency Checks in Practice 64
Reuse 65
Reuse of API Features 65
Reuse of Complete APIs 66
Reusing own APIs 66
Reusing Third-Party APIs 67
Customization 68
Approach 70
Summary 71

Discoverability 71
Manual Discovery 72
Automated Discovery 72
Change Management and Versioning 74
The Evolution Challenge 74
Why does the Evolution Challenge exist at all? 75
Classifying API Evolution 75
Dealing with Evolution in APIs 78
Anticipating and Avoiding Evolution 79
Prevent Feature Creep 80

API Proxy Architecture 83
Requirements for APIs 83
Responsibilities of APIs 84
Desirable Properties of APIs 85
Architectural Patterns 88
Client Server Patterns 88
Facade Pattern 90
Proxy Pattern 91
Architectural Styles 91
REST Style 92
HATEOAS Style 96
RPC Style 97
SOAP Style 99
Architectural Trade-offs 99

API Description Languages 105
What are API Description Languages? 106
API Description Language vs. API Development Language 107
Usage 108
Communication and Documentation 108
Design Repository 110
Contract Negotiation 111
API Implementation 112
Client Implementation 113
Discovery 113

Simulation 114
Language Features 114
Swagger 117
Introduction 117
Example 118
Root Element 120
Resources 122
Parameters 126
Reusable Elements 128
Security 129
RAML 133
Introduction 133
Example 134
Root Element 136
Resources 137
Schema 138
Parameters 139
Reusable Elements 143
Security 145
Summary 147
API Methodology 151
Foundations 151
Consumer-oriented Design Approach 152
Contract First Design Approach 154
Agile Design Approach 154
Simulation-based Design 155
Requirements for an API Methodology 156
Methodology 158
Overview 158
Phase 1: Domain Analysis 159
Phase 2: Architectural Design 162
Phase 3: Prototyping 164
Phase 4: Implementation for Production 167
Phase 5: Publish 168

Maintenance 171
Discussion 172
Hand-over Points 172
Pre-Work vs. Actual Work 173
Summary 174
Conclusion 177
Backmatter 181
Feedback 181
About the Author 181
Other Products by the Author 182
OAuth 2.0: API Security Book 182
API Design Book 183
OAuth 2.0 Online Course 184
References 187
Image Sources 190

Introduction

What is an API?

Software is typically used by people like you and me via a user interface. Increasingly, however, software is not only used by people, but also by other software applications. This requires another type of interface, an Application Programming Interface, in short API.

APIs offer a simple way for connecting to, integrating with and extending a software system. More precisely, APIs are used for building distributed software systems, whose components are loosely coupled. The APIs studied here are web-APIs, which are realized as web services and deliver data resources via a web technology stack. Typical applications using APIs are mobile apps, cloud apps, web applications or smart devices.

The charm of APIs is that they are simple, clean, clear and approachable. They provide a reusable interface that different applications can connect to easily. However, APIs do not offer a user interface, they are usually not visible on the surface and typically no end user will directly interact with them. Instead, APIs operate under the hood and are only directly called by other applications. APIs are used for machine to machine communication and for the integration of two or more software systems.

The only people interacting with APIs directly are the developers creating applications or solutions with the APIs. This is why APIs need to be built with the developers in mind, who will integrate the APIs into new applications. This insight explains, why a new perspective is required for building APIs.

Why APIs?

An API offers a simple way for connecting to, integrating with and extending software systems. Now, think about the entities that are run by software. Businesses, markets and banks are run by software. Industrial production processes are controlled by software. Machines, cars and many consumer products contain software. However, these software systems are typically isolated and functionality of one system cannot be accessed from the other system. APIs provide a possibility to connect these separate software entities.

APIs provide the capabilities which are essential for connecting, extending and integrating software. And by connecting software, APIs connect businesses with other businesses, businesses with their products, services with products or products directly with other products.

The infrastructure for enabling this connection is already in place. Each and every person, each employee and each customer has a smart, internet-enabled device, businesses have websites and web-services. Even an increasing number of the products sold by the businesses carry digital sensors and are internet enabled. All these devices are connected to the internet and can - in principle - be connected via APIs.

Just one example for the business to business integration: The business of an enterprise can be expanded by linking the business to partners up and down the value chain. Since businesses are run by IT, the businesses can be better linked by integrating the IT systems of a business up and down the value chain to the IT systems of other businesses, partners, employees and to customers. This can be accomplished if the IT systems of the business partners are linked via services.

An enterprise cannot force its business partners to use its services. But it can make these services so good -- so valuable and simple -- that the business partners will want to use them. If these services are good, they can become a means for retaining existing partners and a means for obtaining new partners.

But what makes a service good? In this context a service is good if

- it is valuable and helps the partners perform their business.
- it fits the exact needs of the partners.
- it is simple to understand.
- it is easy to integrate and monitor for the partners.
- it is secure, reliable and performant.

Generally, APIs are services that deliver several advantageous properties. This is why they are used for both external integration with business partners and for internal integration within the company. Amazon, for example, uses APIs internally, to integrate the IT systems of its departments. If the interfaces and technology are already in place for internal integration, it becomes easier to provide external integration. External integration is used with business partners or external entities. External APIs are also necessary for realizing mobile apps. Interesting mobile apps use company data, data that is delivered to the app via APIs.

Another reason for using APIs is their use as an innovation lab of the enterprise. To fulfill this vision, the API portfolio should enable the enterprise to build innovative apps with little effort and spark creativity. By making company assets easily available through API, new uses of these assets can be found. Since APIs provide a new, simple way for accessing company assets, assets can be used in new ways within the company. Providing external access to company assets, enables third party developers, who are not even on your pay roll, to create innovations for your organization.

How are APIs used?

APIs are not called by end-users directly. Instead, APIs are called by apps, such as mobile apps, web apps or TV apps. The apps are then offered to end users. The complete solution, which uses APIs, typically consists of:

- A client or app that calls the APIs and processes the data provided by the APIs. This client is responsible for the end-user experience.
- A number of APIs that provide the data to the app.
- An API platform that manages the APIs.

How to build APIs?

I will get back to APIs in a moment. For now, let us assume that we were in the car manufacturing business and we would like to build a new car... What would we have to do?

1. We find out, how the consumer would want to use the new car.

2. We design the car, so it fits into the portfolio of different models that our company sells - sports cars, vans and trucks.

3. We choose the architectural style, i.e. if the car uses a diesel engine, hybrid engine or a fully electric engine.

4. We design a blueprint of the car according to the consumer's needs and wants. We simulate components of the car and build a prototype.

5. We select the component suppliers of our car parts.

6. Finally, we configure the assembly line for putting all the car parts together efficiently.

Could work. And what would the corresponding steps be, when building an API?

1. We find out, how the majority of consumers would want to use the new API.

2. We design the API, so it fits into the portfolio of different APIs that our company offers.

3. We choose the architectural style, i.e. if the API applies a REST, RPC or SOAP style.

4. We design a blueprint of the API using an API description language, such as RAML or Swagger. We simulate the API and build a prototype of the API.

5. We select the API platform, which provides the reusable building blocks for the APIs.

6. Finally, we use a generative API methodology to develop APIs efficiently. Of course, the generative techniques are only used as far as possible, at some point some code might still need to be written.

What is API Architecture?

What most API design books focus on is the use of HTTP methods, URI design, HTTP status codes, HTTP headers and the structure of the resources in the HTTP body. However, this is actually the smallest challenge when building APIs. The real challenge is finding an API architecture and defining the methodology.

API architecture is way more than the correct application of REST principles. So what is API architecture? API Architecture spans the bigger picture of APIs and can be seen from several perspectives:

API architecture may refer to the architecture of the complete solution, consisting not only of the API itself, but also of an API client such as a mobile app and several other components. API solution architecture explains the components and their relations within the software solution.

API architecture may refer to the technical architecture of the API platform. An API platform provides an infrastructure for developing, running and managing APIs.

API architecture may refer to the architecture of the API portfolio. When building, running and exposing not only one, but several APIs, it becomes clear that certain building blocks of the API, runtime functionality and management functionality for the API need to be used over and over again. The API portfolio contains all APIs of the enterprise and needs to be managed like its product. API portfolio architecture analyzes the functionality of the API and organizes, manages and reuses the APIs.

API architecture may refer to the design decisions for a particular API proxy. To document the design decisions, API description languages are used (RAML and Swagger).

This book covers all of the above perspectives on API architecture. Which one are you interested in? Jump to the respective chapter.

How to put API Architecture into Practice?

To become useful, the API architecture needs to be put into practice. This is why this book covers an API methodology for design and development. An API methodology provides practical guidelines and explains how to develop an API architecture into an API that consumers love.

The methodology we propose is an outside-in approach, which also incorporates ideas of contract first design and simulation. In this methodology, the contract is expressed in the form of an API description.

In each phase of the methodology, an API description is either created, refined or used: the API description is the red thread connecting all the steps of the methodology.

Why is API Architecture Important?

It is very hard to move the pillar of a bridge, which is made of steel and concrete. Such changes are difficult, costly and time intensive. This is why a blueprint is created before building the bridge. It allows planning all the details, iterating over several proposals and performing what-if analysis. Changes to the plan are easy and cheap to perform. And by making changes to the plan, it hopefully becomes unnecessary to make changes to the real artifacts. The same is true for APIs.

When APIs have already been built, changes are difficult, expensive and time-intensive. Even worse, the changes to published APIs might break any clients using the API. The consumers might get upset and switch the API provider. To avoid this, the API needs to be right from the start, by the first time it is published.

This can be achieved by planning ahead with an API architecture. An appropriate API architecture increases the efficiency of building the right API, reduces the cost and time for both construction and maintenance and thus reduces technical risk associated with the construction.

An API architecture is an approach for risk mitigation. It enforces that the approach is well thought out before construction is started. It avoids situations, in which resources are spent on implementing APIs, which cannot possibly fly.

An appropriate API architecture enables a contract-first design approach. Once the architecture is externalized and written down, it can be used not only by the API providers to implement the API proxy, but also by API consumers to build apps with this API. The API consumer does not have to wait for the API to be finished, but development of API and app can proceed in parallel.

Non-functional properties of the API should not be an afterthought. The API needs to be designed right from the start to fulfill all non-functional properties such as security, performance, availability. Based on an architecture, the implications of the architectural choices on non-functional properties can be determined early in the design.

Proper architecture and design of the APIs is an investment. In the long run, it will save time and even help avoiding mistakes.