# ACHIEVING ENTERPRISE AGILITY WITH MICROSERVICES AND API MANAGEMENT

JUNE 2016 ■ BY MANFRED BORTENSCHLAGER

A microservices architecture and its underlying APIs allow organizations to build and maintain more agile and flexible IT systems. Microservices support more modular, autonomous systems, and well-managed APIs form the foundation of this architectural framework.

3scale by Red Hat®

# Table of Contents

# Microservices alone do not provide enterprise agility

Microservices have emerged as an architectural style to build and operate IT systems for massive scale based on reusable and replaceable components. Microservices architecture (MSA) uses application programming interfaces (APIs) to connect the various microservices together. Today's speed of business requires IT systems be agile to keep up with rapid pace of change. MSA alone does not deliver that. In part because they require sophisticated ways to manage APIs, and API management solutions address exactly that.

The combined benefits of MSA and API management include:

- Fine-grained and flexible access control to microservices
- Microservices usage analytics
- Configuration of usage contracts such as rate limits
- Automatic and interactive interface documentation
- Adaptive microservices description and discovery

This paper focuses specifically on how API management features can be deployed to achieve enterprise agility with microservices.

The main idea behind microservices is to break modules down into "small enough" components but not smaller. James Lewis and Martin Fowler provide a more formal definition:

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and are independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

In this article we will not discuss definitions or benefits of microservices in detail. There are many excellent works available that cover this such as Sam Newman's book on Building Microservices or Martin Fowler's foundational article on Microservices.

# Business and technology challenges

Coined by R. Buckminster Fuller, the "ephemeralization" of technological advances – or the ability to do "more and more with less and less" – results in the increased speed of business as well as many business and technology challenges. Microservices architecture in combination with API management provides an ideal architectural style to address these challenges.

## Constant and rapid change

Today's business world is characterized by constant and rapid change in customer demands, new competition, and evolving technology. The typical monolithic approaches to system design and even sophisticated layered architectures are too tightly coupled, and engineering teams can't keep up with system changes sufficient for the rate of the market changes. However, with a microservices approach, this is now possible due to the most fundamental aspect of a microservice: its autonomy. Autonomy results in decoupling, and decoupled system artifacts make them easy to change, making microservices better able to cope with business changes.

## Omni-channel strategy

The fact that mobile is becoming an equally important channel to Web (in some cases even more important) is persuading many business leaders to entertain several channels to market. The Internet of Things is probably the next big channel trend, although it is in fact a collection of channels itself – cars will have to be served differently than home automation appliances. Building silos for each channel will not be scalable or effective. Microservices help to break these silos.

## Modularization

The key technical challenge to forming microservices is to break up larger systems into their smaller components. The key questions become about figuring out the right components and their size. This often depends on the context of an organization. Another challenge is how to coordinate the many frequently changing microservices. A service discovery is one answer to this question, but it poses other challenges at the same time.

## Security

With so many constantly changing microservices making up a system architecture, how can you consistently provide the required level of security? API management helps to keep everything under control and also allows different levels of security, as some services may be more mission-critical than others. External-facing services also require different security mechanisms than internal ones.

## Service discovery

A consequence of the autonomy and decoupling characteristic of microservices is a shift of complexity. Services themselves become smaller, focusing on doing one thing only – but very well – and in this way become less complex. However the service description, discovery, and aggregation become more crucial and complex. If this "glue" between microservices doesn't work, the whole MSA won't work. We present an API management-based microservice description and discovery approach.

# Microservices architecture with API management

HTTP-based APIs are usually central to a microservices architecture. APIs are the glue that connect the various microservices together. So having a good way of designing and managing the multitude of APIs is crucial for an effective MSA. The use of API management advances MSA to achieve true enterprise agility.

## API design for microservices

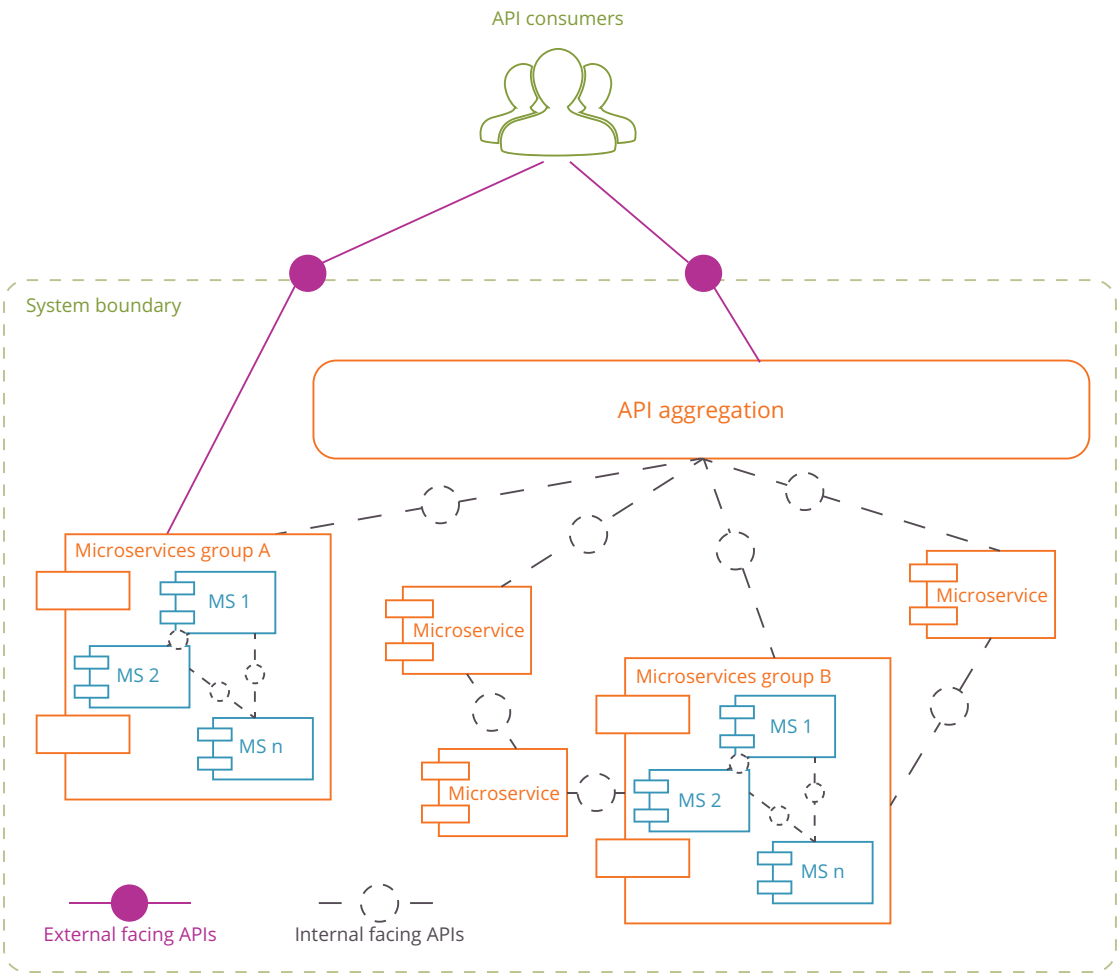
API design and microservices design go hand in hand. It doesn't matter which you choose to design first– the API or the logic for the microservices and underlying data resources – you will iterate between the two several times. What's important is that your API and microservices design depend on two aspects: your specific business domain and the needs of your API consumers. Understanding both is essential for a good and sustainable design. With that background, we at 3scale always recommend designing for simplicity and flexibility.

What we mean by simplicity: you have to understand the most common and most valuable use cases of your API or microservices consumers. These use cases must be supported by your API design as first-class citizens. On the other hand, the API design must be flexible enough to support uncommon, unexpected, or unforeseeable cases. It's important not to exclude interesting cases in order to allow for innovation.

Simplicity and flexibility are often achieved through fine-grained vs. coarse-grained API design and aggregations. If you want to dive deeper into the specifics of API design, we recommend checking out the API Owner's Manual, which includes a whole chapter dedicated to this topic.

## Internal and external APIs

Microservices communicate through APIs. There is an important distinction between APIs for microservices that are invoked within system boundaries (internal APIs) and API calls that are issued from outside (external APIs). All of these APIs need to be managed but depending on the business and architectural context there may be differences between API management for internal vs external APIs.  The diagram below describes a MSA with the various communication paths between microservices enabled through APIs. External APIs are indicated with solid circles. The internal APIs on the system boundary are segmented.




Microservice-based architecture with internal and external APIs

This diagram shows a couple other important concepts. Within the system boundary, there are a number of individual microservices. In addition, there are microservice groups or modules. These are cohesive sets of microservices that together form a specific higher level service. For example, an online shop service might be a higher level service that requires several downstream services such as a payment service, shopping cart service, wish list service, recommendation service, and more.

## Orchestration and choreography of microservices

Microservices in a set are usually either orchestrated or choreographed. Orchestrated means that there is a central entity (ideally a microservice itself) that coordinates communication between various services. Whereas for choreographed microservices, the various microservices manage themselves. The orchestrated approach is easier to achieve. However, it results in dependence and a single-point-of-failure for the one coordinating service. Both have their pros and cons, and how you design your MSA depends on your particular context.

## Aggregation layer aids in downstream service management

Another effective concept shown in the diagram is the API aggregation layer, shown at the top. As the name suggests, this layer aggregates various downstream services and microservice groups into the services that are then exposed to API consumers. This ties into the concept of designing APIs for simplicity, as the aggregation layer combines the services and provides APIs according to the most common use cases. This approach also makes it easier to maintain or change the downstream services because there is another abstraction layer and no direct access to the microservice. An example of design for flexibility is the second external API, which connects directly the microservice group A. How simple or flexible the APIs in your MSA are depends, again, on your particular context – your business domain and your API consumers' needs.

The Backend for Frontend (BFF) pattern credited to Phil Calçado is a refinement of the API aggregation layer. In simple terms, BFF is a feature-specific facade that sits on the server side and provides APIs customized and optimized for certain types of clients and their platforms, such as web browsers or mobile apps. The idea is to have several BFFs, depending on how many different such channels should be supported and how different the clients are. With this pattern, omni-channel strategies can be implemented more effectively. Ideally, BFFs are microservices themselves.

# How API management solves the agility problem of microservices

We just discussed the differences between internal and external APIs. The distinctions have important consequences in how API management is used to ideally leverage each case. Now we move on to describe some key API management features and how they are deployed differently for internal vs. external APIs in MSAs to deliver enterprise agility.

## Security

Security mechanisms differ for different types of APIs. For less mission-critical internal APIs, simple protection with API keys is usually sufficient. For external or critical APIs, a more secure approach like OAuth will be required.

## Usage contracts

Good API management solutions also allow you to define usage contracts based on metrics such as number of API calls. Internally, you can handle this more liberally, but control may be tighter for external API consumers. In addition, API consumers can be segmented and differentiated access tiers, and service quality can be offered to different segments.

## Documentation

Having API management in place for an MSA also makes it easier for API consumers to find and understand how to use the APIs. Having a developer portal is a common best-practice, which supports typical developer onboarding processes like signup and account administration, especially for external API consumers. Developer portals typically also provide API documentation. The best-of-breed even allow interactive access to the live microservices through an API. This allows developers to get a good impression about the API's behavior, without needing to write one single line of code.

## Monetization

API management can also be used to monetize access to the microservices behind the APIs. This would be relevant for scenarios involving external API consumers.

## Analytics

What all APIs – internal as well as external –will have in common is the need for analytics, reporting, and alerts. It's essential to know what's going on with the APIs – which consumer or app is calling which API and how often. It's also important to know how many APIs fail, due to what reasons, and to have latency information. These are important metrics that allow a microservices provider to understand system behavior better in order to optimize it and make better decisions.

## Discovery

MSA can address the rapid change in the business world better than more traditional approaches to systems architecture. This is due to the fact that microservices themselves are very dynamic, which is achieved by decoupling to reduce dependencies. API management provides the required discovery mechanisms to make sure that available microservices can be found and information is shared about how to use them.

# Sample implementation of a combined microservices architecture with API management

This section covers how to achieve a MSA with API management using the 3scale API Management Platform.

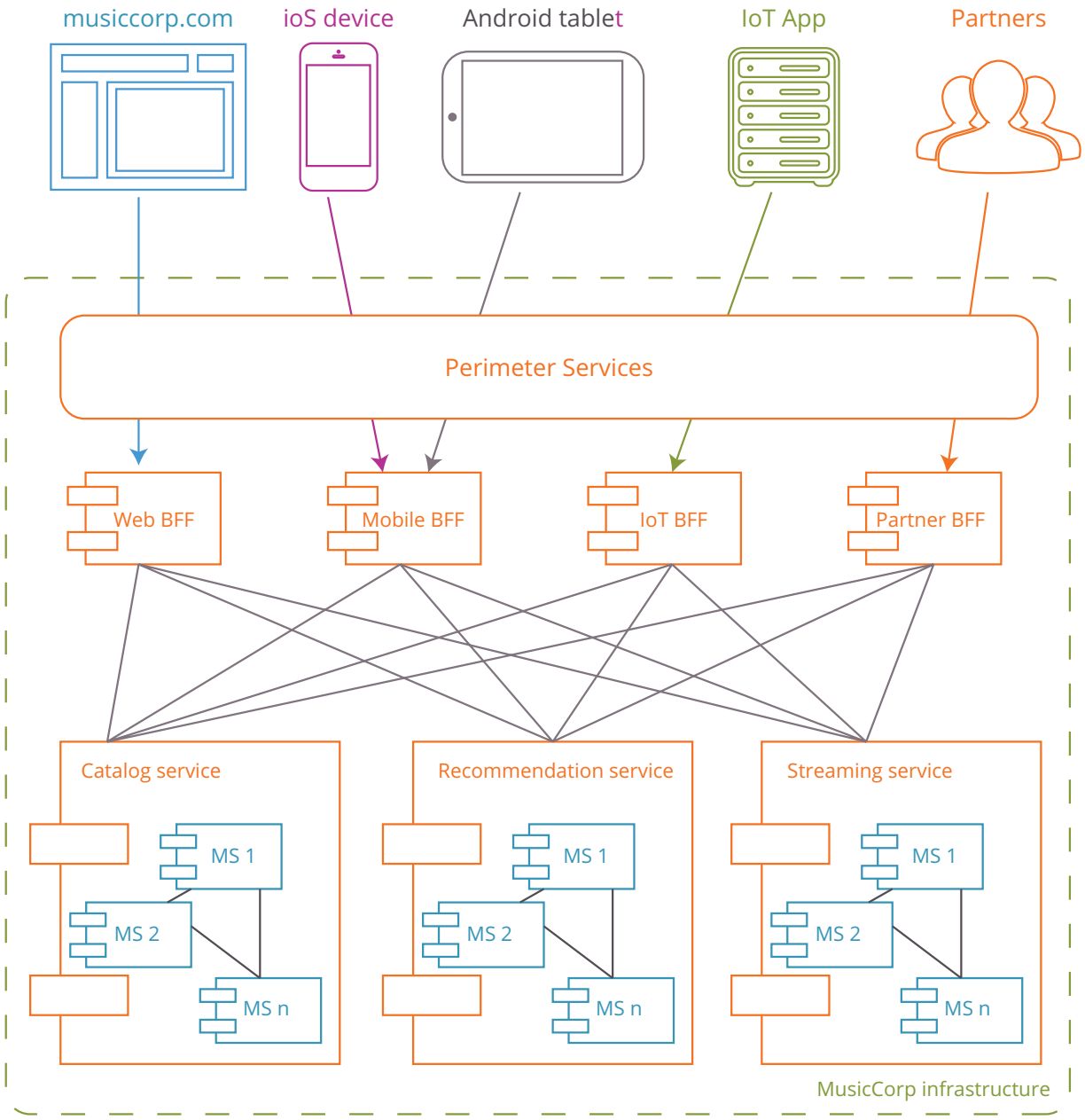
In order to make it clear, we're borrowing a fictitious example - MusicCorp - from Sam Newman's book Building Microservices. We use it here to show microservices and API management together in action. Here is how Newman describes MusicCorp, their model, and strategy:

> *[MusicCorp] was recently a brick-and-mortar retailer, but after the bottom dropped out of the gramophone record business it focused more and more of its efforts online. The company has a website, but feels that now is the time to double-down on the online world. [...] Despite being a little behind the curve, MusicCorp has grand ambitions. Luckily, it has decided that its best chance of taking over the world is by making sure it can make changes as easily as possible. Microservices for the win!*

Using an MSA approach and incorporating concepts we've previously covered, we model several MusicCorp services, including:

- Catalog service
- Recommendation service
- Streaming service

The following diagram depicts an extract of the MusicCorp example architecture showing these services. This also includes a lot of internal and external APIs, all of which need to be secured and monitored using API management.

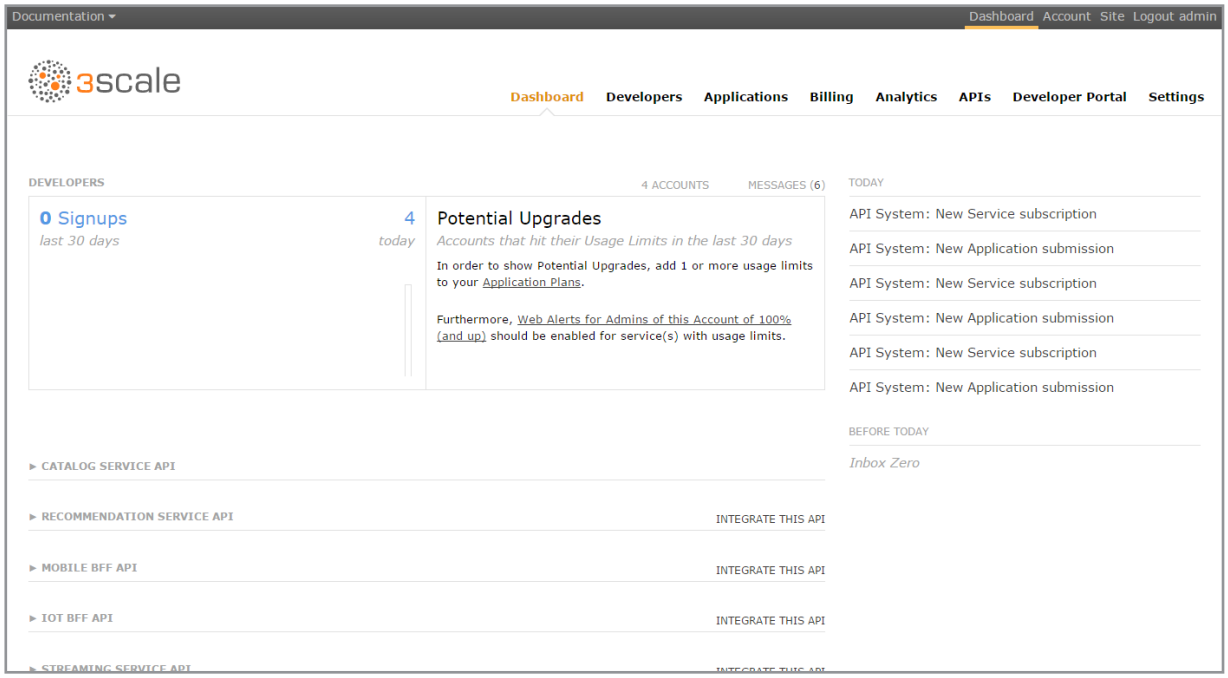



Our rendering of MusicCorp's infrastructure model

# Modeling MusicCorp with API management

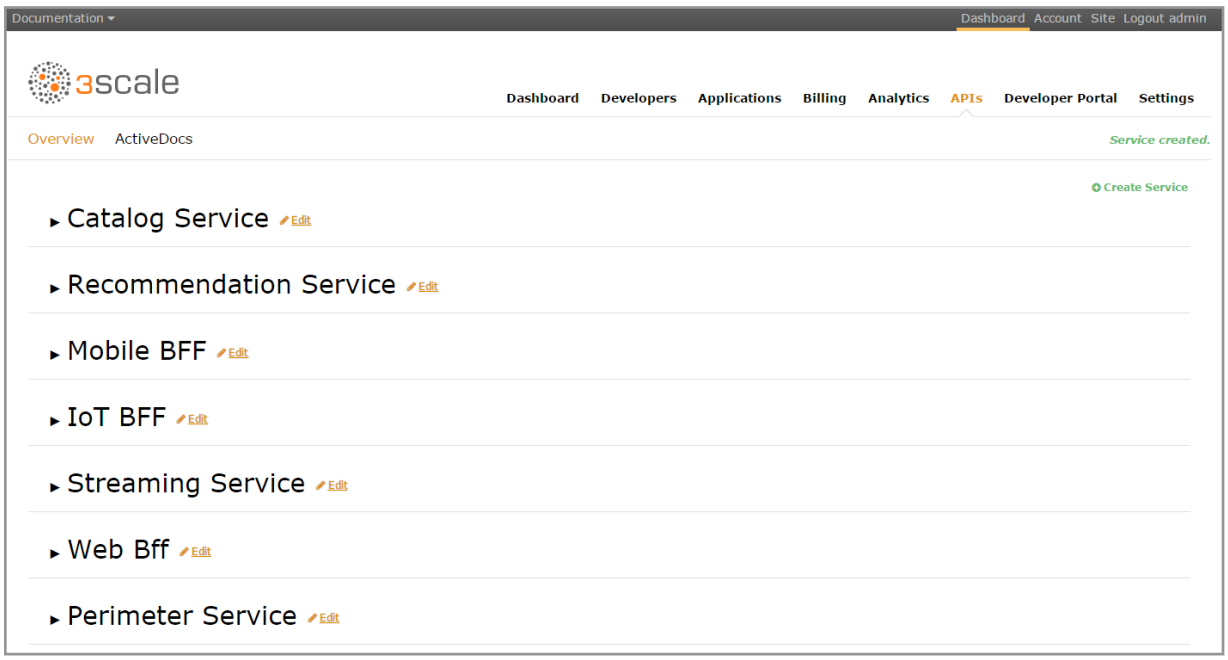
3scale offers a full-featured API management solution with features including access control and security, defining API contracts (like rate limiting), API analytics, developer portals with interactive API documentation, and monetization. We will use this to show how we can manage MusicCorp's internal and external APIs.

Here the main microservices for this example are modeled in the 3scale Admin Portal. After the API management administrator logs in, they are presented with a comprehensive dashboard that provides the key performance data in one view. For example, it shows the new API signups per day in the top left-hand corner.



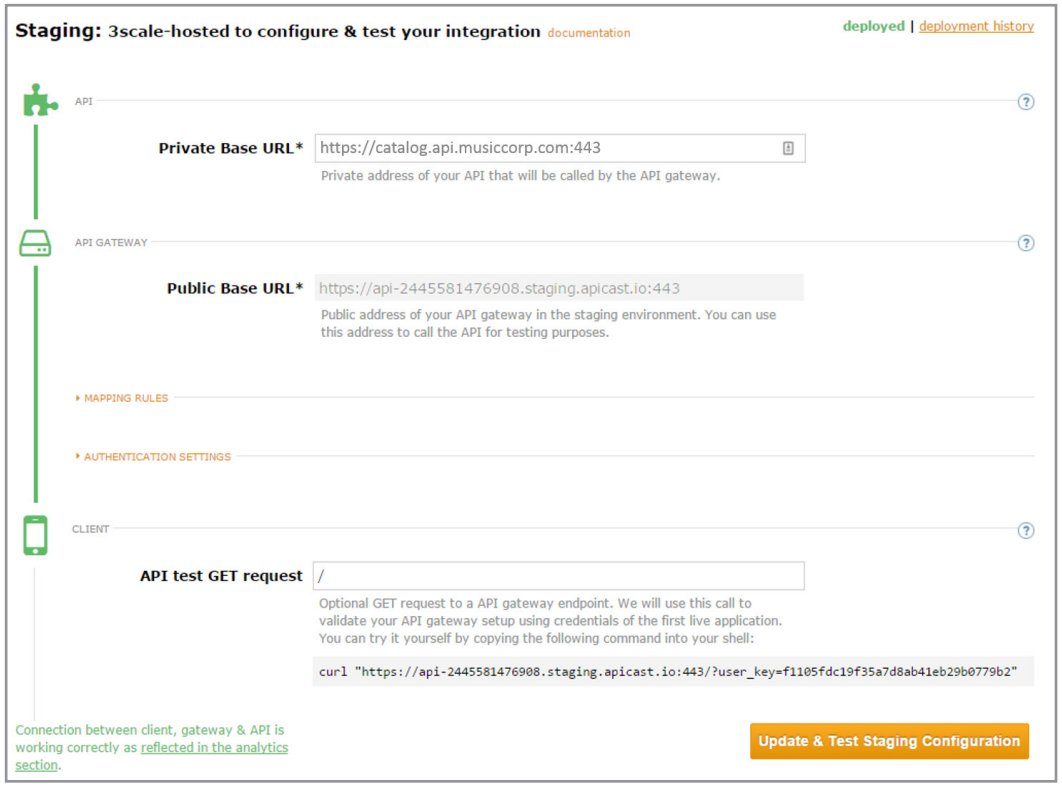

3scale API management dashboard

Under the "API" tab on the top navigation bar, you can see the various APIs of the microservices configured in 3scale. The listing includes internal as well as external APIs and we can define the API management policies individually for each.



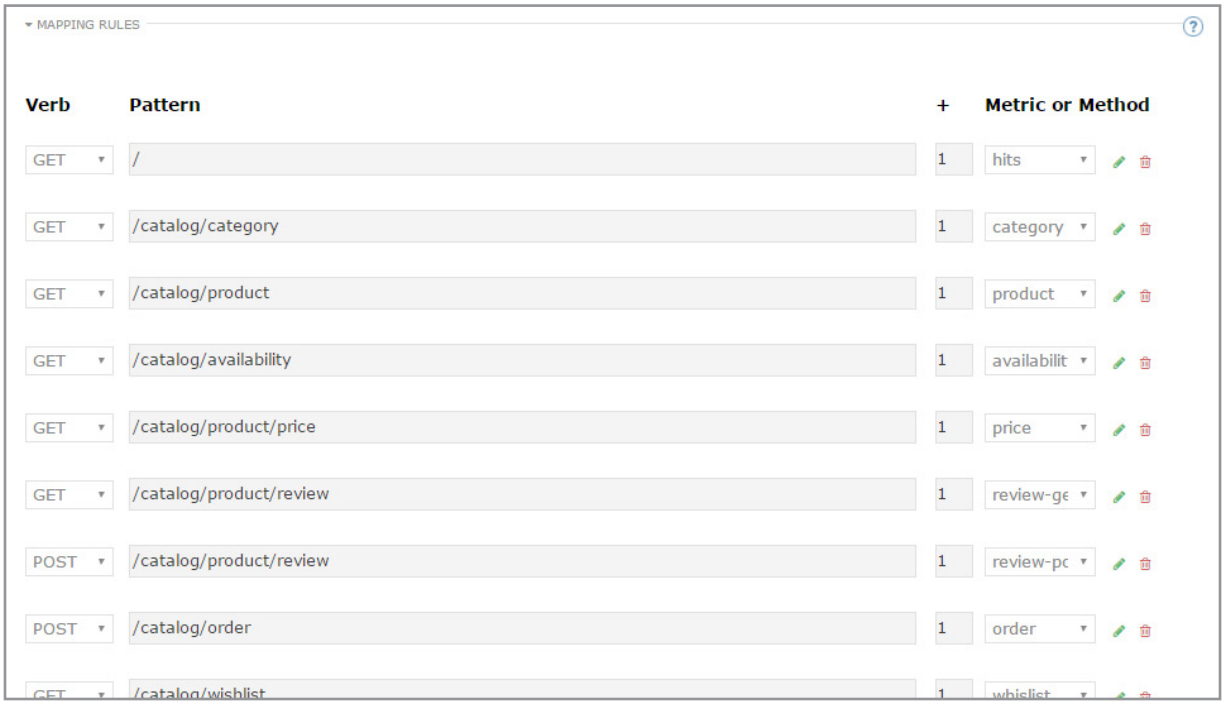

List of MusicCorp's API managed microservices

There are more fine-grained API service operations and features for managing microservices, as you can see using the catalog service as an example.




Integration of a microservice with 3scale API management

In the mapping rules section, you can see the various API endpoints that represent the microservices that belong to the catalog service microservice group.




List of HTTP endpoints representing microservices

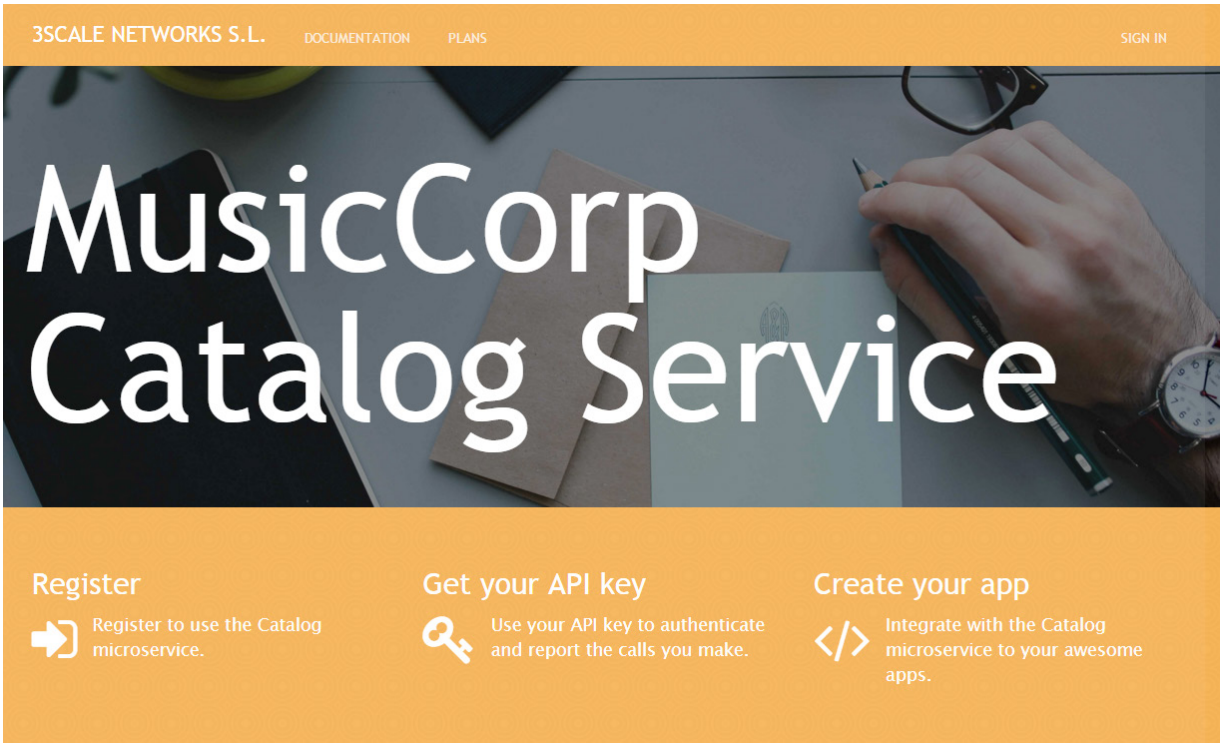
Having visibility into when and by which user each microservice is used is an essential feature. 3scale offers API analytics for that under the "Analytics" tab. There, you can again choose the catalog service to get some insight into how this service is used.

In the chart below, you can see the overall hits accumulated for the whole microservice group, and you can also drill down into the API endpoint of each microservice involved.


Analytics of microservice usage (total number of all service invocations


Analytics of microservice usage (selection of specific service)

With 3scale, you can also expose these microservices' APIs to developers and provide interactive API documentation. The Developer Portal feature is particularly useful when it comes to external or public APIs. In this MusicCorp example, the catalog service is exposed to internal developers because all the various microservice groups and BFFs are developed and maintained by different teams. The screenshot below shows this Developer Portal. It's a one-stop-shop to sign up, get the necessary API keys, manage applications, and get to know the behavior of this microservice by using interactive API documentation.




Developer portal for MusicCorp's Catalog Service

# Microservice description and discovery

As outlined previously, description and discovery of services is essential for providing enterprise agility, especially since the parts of a MSA are constantly changing. In order to address this, we recommend having a dynamic microservice search engine deployed, which makes it easier for developers to discover API descriptions that point to the microservices. We recommend the free and open-source APIs.io search engine.

APIs.io is a public API search engine. However, it can also be deployed on premise and used internally to describe, publish, and discover APIs for microservices automatically. You can get the source code from GitHub.

At the core of APIs.io is the APIs.json description format. APIs.json is a machine-readable approach that microservices developers can use to describe the APIs to their microservices, similar to how websites are described using sitemap.xml. There is also an APIs.json editor available here. Once this is done, all you have to do is point APIs.io to your APIs.json.

The image below shows the beginning of how the MusicCorp catalog service would be described in APIs.json.

```
{
  "name": "MusicCorp Catalog Service",
  "description": "This is a MusicCorp internal microservice group that encapsulates all functionality necessary for a product catalog.",
  "url": "https://catalog.api.musiccrop.com",
  "image": "https://catalog.api.musiccrop.com/logo",
  "specificationVersion": "0.14",
  "apis": [
    {
      "name": "GET category",
      "description": "",
      "image": "",
      "baseURL": "https://catalog.api.musiccrop.com",
      "humanURL": "https://catalog.api.musiccrop.com",
      "properties": [
        {
```

APIs.json service description for MusicCorp's Catalog Service

After that, this microservice is discoverable through APIs.io as a central search engine. Whenever the microservice API changes, this change is reflected in APIs.io too (as long as the APIs.json is updated, which can be automated). All of the functionality of APIs.io is also available through APIs, which allow you to integrate it into your system for DevOps and automation.

# Conclusion

Microservices architecture in combination with API management is a useful approach to deal with today's speed of change in business. Although they have many advantages, microservices are not a silver bullet and have their own challenges that need to be addressed. Since microservices are inherently related to APIs as their main communication and interaction mechanism, solid API management solutions improve microservices architectures and deliver on the promise of achieving enterprise agility.

The MusicCorp example specifically shows how to model an MSA using the 3scale API Management Platform. 3scale provides key features such as different levels of access control and security for different types of microservices (internal vs. external), definition of different segments of microservices and their API consumers in order to serve them differently, and API traffic analytics and reporting. Microservices APIs can also be exposed through developer portals and include interactive API documentation. Finally, APIs.io and APIs.json provide a solution to microservice description and discovery.

Ready to embark on your MSA and API management journey? Sign-up for a 3scale account and try it for yourself. Or you can get in touch with us to discuss your goals and approach.