

Accelerating OpenAPI Design & Development with Next-Level Mocking

Content

Introduction to OAS-Driven Development.....	4
Basic Concepts: Mocking and Sandboxing.....	5
Four Challenges with Mocking and Sandboxing.....	6
Getting Started with Basic Mocking.....	8
Taking Basic Mocking to the Next-Level with Service Virtualization.....	10
How Does Virtualization Compare to Mocking?.....	12
API Virtualization with ServiceV Pro.....	15

Introduction

What will we cover in this eBook?

We'll start by explaining the role of the OpenAPI Specification (OAS) in API development. Then we will cover what mocking and sandboxing mean, and then how you can take an OpenAPI definition and create a mock out of it for different purposes. Next, we'll take a look at service virtualization: what it is, how it's different than mocking, and how it can be incredibly beneficial to you and your team when designing and developing APIs.

By the end of this eBook, you'll be able to understand how easy it is to take an OpenAPI definition (or any API definition), pull it into a tool, and quickly create a next-level API mock using a technique called virtualization. From there, you can easily share that next-level API mock, or virtual API, with internal or external teams – all without doing any coding.

Editor's Note

Throughout this eBook, we'll use the terms service virtualization and API virtualization interchangeably, as, for all descriptions and use cases we discuss, there is no major difference between them.

Introduction to OAS-Driven Development

The OpenAPI Specification (OAS) has emerged as the de facto standard for defining RESTful APIs. According to the State of API 2019 Report, 69 percent of organizations are developing APIs with OAS. This number is up from just 25 percent in 2016.

The growth in adoption of standards like OAS has led to a movement to a design-first approach to API development. In a design-first approach, teams start with the design of an API, getting alignment earlier in the lifecycle before a sign line of code is written. The definition acts as the blueprint of your API and can be leveraged to automate tasks throughout the API lifecycle — including development, testing, documentation, and monitoring — and can streamline the development workflow across teams.

But while this approach has benefits, it can often be difficult to migrate to because of the risk of development bottlenecks and dependencies on different teams.

This is where having the ability to generate mock or virtual services, based on the OAS definition, becomes important. With tools like SwaggerHub and ServiceV Pro, teams can collaborate on the design of the API, and virtualize your API definition to start testing and developing in parallel.

With the ability to setup realistic, dynamic mocks in seconds from your existing OAS definition, you can easily remove the bottlenecks in your development workflow and make the move to a design-first approach with OAS.


In the next section, we will look at the different approaches to mocking APIs, and the benefits of implementing a virtualization tool.

Basic Concepts: Mocking and Sandboxing

When you're developing or testing APIs, there are many cases where you'd want to see the behavior of the API before it's fully built. Two popular ways of doing this are through mocking and sandboxing.

A mock is a static emulation of an API. You may not have an actual API yet, but you might want to make a clone of that API for your own purposes. You'd want that mock, or clone, to return a response when you make a request. To create a mock version of an API, you might write code, or use an open source tool like SoapUI. Sandboxing is another way to do this. A sandbox is essentially a pre-provisioned instance of an API endpoint, created by a third party. For example, if you're working with a bank, that bank might give you a sandbox that you could use for your purposes of development and testing.

The issue with mocks and sandboxes is that they are frequently controlled by a third party (i.e. they are out of your control), and each is only valid for a very specific use case – making one request, and returning the same, single response every time. We'll talk about these limitations more in the next section.

	Mocks	Sandboxes
Definition	Static emulations of APIs that are generally coded in a programming language	Pre-provisioned instances of API endpoints, often provided by a third-party for specific use-cases
Tooling	 SoapUI { ... } SWAGGERhub	API service providers like PayPal

Four Challenges With Mocking and Sandboxing

While **mocks** and **sandboxes** are useful in certain cases, there are a lot of issues that you might encounter when working with them. We've outlined the top four:

1 They're very hard to create.

Mocks need quite a lot of manual configuration. You have to actually do a lot of scripting to create a mock from an API. So, often what's going to happen is that you're going to ask your developer, who has developed the API, to create these mocks for you, so you depend

on other teams to create these mocks and sandboxes. Unless you're a developer of the API yourself, it would be very hard for you to create these mocks.

2 They are owned by other teams and third parties

You may require certain use cases from these APIs, but it would be very hard for you to convey that information to the third party or other team and ask them to change it according to your specifications, because of course those people have their own timelines and deadlines; they may or may not comply with your request.

3 They are limited to very few scenarios

Even if you're doing mocking yourself, you can do it for one particular scenario. That means, for example, you can do it for a login. But then, if you want to take that existing scenario and change it to something else, it requires a lot of effort. So, it's very hard for you to create multiple responses on these mocks and sandboxes.

4 You cannot attach a state to mocks

By "State" we mean credential information or cookie information that you want to take and pass throughout different responses of your API mocks. So, it's very hard to do it. Take the example below: if you're creating a mock from an OpenAPI Specification, the response would look something like what you see in the box. You will have different values in this response, but unless you have a software or a tool that can change these values for different parameters, it will be very hard for you to do it yourself.

Static Mock



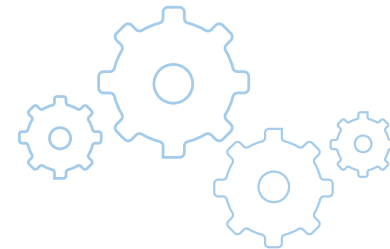
Response

```
{
  "id": "d290f1ee",
  "name": "ÖText Widget",
  "manufacturer": {
    "name": "ACME Corporation",
    "homePage": "https://www.acme-corp.com",
    "phone": "408-867-5309"
  }
}
```

- ✗ Multiple scenarios
- ✗ Routing between scenarios?
- ✗ State-full scenarios?

The name, in this example, is 'ACME Corporation'. However, you may want to test if this API returns a response for another company name in another language, like Mandarin or Spanish. Unless you have a tool that enables you to change that name quickly, doing so yourself is going to cost you a lot of time and effort as you try to scale. You could do it for five or ten company names, but what happens if you have to test that API for maybe a 1,000 different scenarios? You'd run out of time to do it.

To summarize, mocks and sandboxes cannot cover multiple scenarios, it's very hard for you to route between your mocked API and the actual API, so you will have to change the URL in your code or in your testing setup, the URL will have to be changed to point, one time to a mock and then, later on, to an actual API. So, it's very hard for you to do, because it has to be done manually. The mocks and sandboxes do not provide you an attached state. You cannot actually take a set of parameters and parse them across different responses. So, these are some of the problems that you face with mocking and sandboxing.



Getting Started With Basic Mocking

Now that you're up to speed on what the OpenAPI Specification is and how mocking works, let's look at a concrete example. The figure below shows how you can take the OpenAPI definition, and before it's made available to your consumer (which could be a developer or a customer or a user), you have to develop it and make that API available to these consumers after you have coded it. Once you've done that, you can make it available to your testers or third-party developers who want to use it.



From there, you'll have to test that API to ensure that it functions properly and performs under a heavy load of users. Only once the API has been tested and subsequently made available in production, are you able to give it to your end users.

In the case of developing an API for an app, the API might only be ready to be used once it's available in production. From there, you can integrate it with your app, and only then can that app be made available to your end users. This is a long process – so you might try to do some prototyping of your application or APIs. Unfortunately, prototyping also requires a significant amount of time and effort before you can actually make a prototype

version of the API available to your end users. In this case, mocks aren't useful because they are so static – they cover just two or three scenarios, they don't have a state, and as a result, you can't do much with them. Instead, you have to go through with this whole lifecycle, or at the very least a few stages of it, before you have an actual API available to you. To solve that problem, you'd use a technique called service virtualization.



What is Service Virtualization?

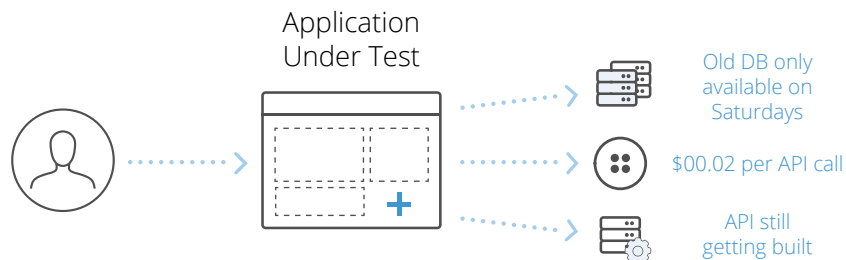
Service virtualization is a way of accurately simulating the behavior of a component that is not available or is difficult to access during the software development lifecycle. Here's an example of how it works:

Say, for instance, you have a component – an API, a device, or a database – which is either not available, really expensive to set up, or is owned by a third-party. This is a very common problem for most development and testing teams. With service virtualization, you could virtualize the unavailable component and make it available for the purposes of testing, development, and prototyping.

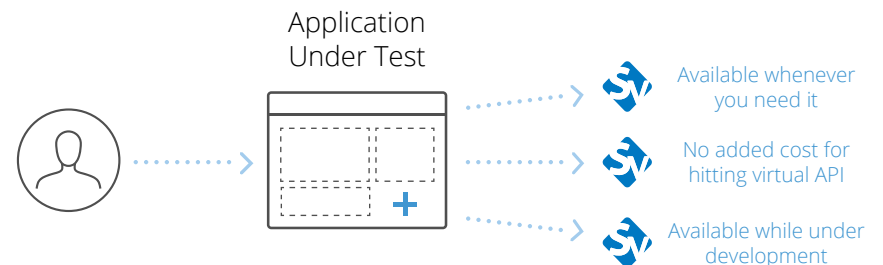


In this eBook, we'll talk primarily around API virtualization. With API virtualization, you can take an API that is not available (because it's owned by a third party or is expensive to invoke – think of the Google Maps API) or is in the initial stages of design and development, and create a very realistic simulation of that API using service virtualization. Conceptually, virtualization allows you to take your basic, single response mocking to the next-level.

Traditional Testing of an Application & Its Dependencies



Testing an Application With ServiceV Virtualizing Dependencies



How Does Virtualization Compare To Mocking?

The problem with mocking, as mentioned previously, is that it's static. You can mock APIs for one or two responses fairly easily, or maybe 10 or 20 responses if you really put a lot of time into it – but that is going to take a ton of time and likely isn't going to cover all your scenarios.

Conversely, with virtualization, you have the ability to actually take one single response of an API or one particular endpoint, and then quickly clone it or link each and every response parameter of a column of a data source. The data source could be a database, or an Excel file,



Want to learn how you can delivery quality applications, faster?

Read our guide on the benefits and use cases of service virtualization

[Read the guide](#)

that contains hundreds and thousands of rows. In a sequence, your virtualized API will respond, one by one, to the values from that data source. So, it enables you to do much more in a very short period of time.

Compare this to mocking, which, again, is stateless in nature, and doesn't enable you to add a state or login credential across multiple requests. So, mocking works fine if you just want to check one API endpoint and see, "Hey, what's going on with this API? Which kind response am I going to get? What does the structure look like? Is it working or not?"

In a different scenario, if you said, "Hey, I want to log in, do a transaction, and log out," this would require three or four API calls, plus you'd have to transmit a lot of cookie and credential information across these API calls. With mocking, this would require 100% manual configuration, which is challenging if you don't know how to code.

Mocking API Calls

- ❌ Static implementation for specific situations
- ❌ State-less, responses do not have contextual information
- ❌ 100% manual configuration
- ❌ No re-use

Virtualizing APIs

- ✅ Full implementation of the service
- ✅ State-ful, Responses are aware of the context
- ✅ No configuration needed
- ✅ Re-usable artifacts

Service virtualization, however, allows you to set these parameters upfront and then parse them as cookies, or headers, across all the requests – with no configuration required. Just set it up, save the project, and share it with your team. You could share the virtual API over Git or share it with teams across the globe over the network or the cloud. Just deploy it on the cloud component, and it will be available for teams – internal or external – to access all over the world.

To take it a step further, you can then reuse that virtual service, by starting with what you initially created, and copying it and multiple scenarios a number of times to do more with your virtualization. This is a much more complicated process with mocking, as it would require you to take all the scripts and copy them all over again, then modify them all over again – a process that takes a lot of work.

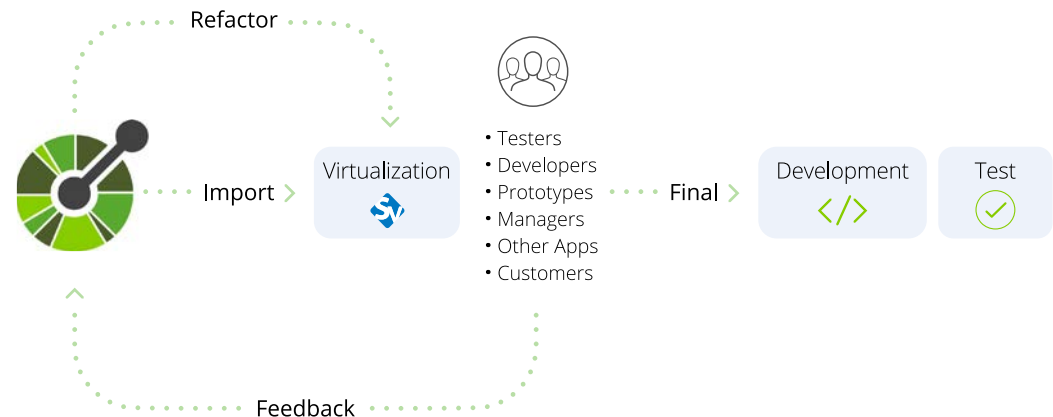
Imagine the amount of work that you would have to put in when you have to change one scenario; then go in and change the code for each and every place where you've copied that script for your mocks. This is an example of where virtualizing your API instead of mocking it dramatically increases efficiency in your delivery lifecycle.



API Virtualization in Servicev Pro

When you're ready to take your mocking to the next level by virtualizing your APIs, ServiceV Pro from SmartBear is a tool that can help you get started. It enables you to quickly create virtual environments for APIs: you can record and replay APIs, modify them, and make them really advanced and realistic. ServiceV Pro can also help you do this with your database (JDBC) transactions. You can easily record, modify, and use your API transactions for testing, development, or prototyping purposes.

Recording is easy too. You can create realistic virtual services by linking them to data, response parameters – each and every parameter – to a column in the database in just a few clicks. Once you've done that, your response of your API will pick values from each and every row of your database and respond that in a sequence or a logic defined by you.



Share your virtual APIs with anyone

Next, you can take your virtual APIs, and share them with anybody. When might you want to do this? Say you're an API developer or designer, and your consumers or your customers want to see what work you've completed -- they might want to see a realistic simulation of your API. You can take your OpenAPI Specification, put it on ServiceV Pro, create those virtualized APIs that are really advanced, and then share them over the network with your customers.

Put your team in control

At the end of the day, service virtualization puts you in control, so you don't have to depend on a third party. It saves you from having to call somebody to make modifications through a sandbox, in order for you to do what you need to do. Service virtualization will also save you a ton of time, so you can do prototyping early -- without all that configuration and manual coding -- so your team is able to deliver products faster to the market.

The time savings and increased control over services will ultimately help your organization's bottom line as well. On top of that, you'll be delivering higher quality products, because you were able to simulate those scenarios upfront and test them and develop them, and you give those APIs to your customers earlier too.



ServiceV Pro

Reduce dependencies on
internal and 3rd party services
with virtualization

Get Started

