O'REILLY®

# Service Virtualization

## Implementation, Practices and Trends for On-Demand Test Environments

Bas Dijkstra

Accelerating next

Hewlett Packard
Enterprise

# Thrive in the new now: Engineering for the Digital age

## Is your application fast enough?

**Get** your custom **HPE Insights performance report NOW** to learn how your application is performing: www.hpe.com/software/insights.

You will receive a detailed performance report in less than 5 minutes.

Hewlett Packard Enterprise software enables you to deliver amazing applications with speed, quality and scale. **Learn more:**

Mobile testing

Web Performance & load testing

Network Performance

Simulate constrained environment

# Service Virtualization

*Implementation, Practices, and Trends for On-Demand Test Environments*

*Bas Dijkstra*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

**Service Virtualization**

by Bas Dijkstra

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

**Editors:** Brian Anderson and Virginia Wilson
**Production Editor:** Colleen Lobner
**Copyeditor:** Octal Publishing, Inc.
**Interior Designer:** David Futato
**Cover Designer:** Randy Comer
**Illustrator:** Rebecca Demarest

October 2016: First Edition

**Revision History for the First Edition**

2016-10-12: First Release

978-1-491-97073-7

[LSI]

# Table of Contents

# Introduction and Reading Guide

Increasingly competitive and rapidly changing markets are forcing organizations that rely on software either as their primary source of revenue or as a critical supporter of their business processes to be able to design, develop, and release high-quality software at speed. Failing to deliver new releases quickly and efficiently, or delivering software that contains too many defects, will have a negative effect on your competitive edge and therefore on revenue.

Also, contrary to traditional monolithic software systems, modern applications consist of an increasingly large number of independent, interconnected components. This building-block approach to software design and development promotes reuse, maintainability, and parallel development.

These two factors place high demands on the software development life cycle and especially on the testing activities incorporated therein. Development teams need to be able to continuously test and release software, which in turn requires that test environments need to be available and ready for testing at all times. Anyone involved in software testing, however, can tell you that managing test environments is no ordinary feat. Having all components and dependencies in place, virtually on demand, takes a lot of time and effort, if it is even feasible at all.

One approach that has seen a steady rise in popularity in recent years is the introduction of *service virtualization* as a means for development teams to regain control over the availability of suitable test environments and, as a result, over their software development life cycle as a whole.

This book provides an overview of the current state and trends in the field of service virtualization. We begin with an introduction of the concept and an overview of the benefits that service virtualization brings to the software development life cycle. Then, we'll take a look at the role that service virtualization plays in bimodal IT and in the introduction and execution of Continuous Delivery. The final part of this book contains insights in the role that service virtualization plays in current IT trends, such as Agile software development, the Internet of Things, and the API economy.

This book is intended for organizations and professionals involved in the software development process that want to inform themselves about service virtualization as a means of improving testing and software delivery processes as well as about the current state of the field and the way service virtualization can add value with regards to upcoming IT trends.

Please be informed that any technical details with regard to tool- and vendor-specific service virtualization implementation strategies are beyond the scope of this book. For more details on these, refer to the website of the corresponding tool or vendor. From personal experience, these usually contain myriad technical information as well as case studies describing implementation strategies in much more detail.

CHAPTER 1

# An Introduction to Service Virtualization

In this chapter, we learn what service virtualization is and how it relates to stubbing and mocking, two other simulation techniques that are used often by development teams to help them write and execute tests effectively.

## What Is Service Virtualization?

> *Service virtualization* is a method to emulate the behavior of specific components in heterogeneous component-based applications such as API-driven applications, cloud-based applications and service-oriented architectures. It is used to provide software development and testing teams access to dependent system components that are needed to exercise an application under test, but are unavailable or difficult to access for development and testing purposes.[1]

A lot of modern software, such as Application Programming Interface (API)–driven or Service-Oriented Architecture (SOA)–based applications, consists of a number of interconnected components. Software development teams that want to access these dependent components (dependencies) during development and testing often find that these dependencies are unavailable or difficult to access.

1 *https://en.wikipedia.org/wiki/Service_virtualization*

There are several reasons for this:

- The dependency has not yet been developed or is under development. This is often seen when several development teams work in parallel on different components of a single system.
- The dependency does not contain appropriate test data. When test environments need to be configured with complex test data structures, often an (anonymized) copy or subset of production data is loaded into the test environment, without the development team knowing the contents of this dataset.
- Access to the dependency requires an access fee. This is often the case with Software as a Service (SaaS)–based third-party dependencies.
- The dependency is otherwise unavailable, unreliable, or acting in a nondeterministic manner. This is often the case with legacy systems.

Service virtualization attempts to remove these test environment constraints by simulating the *behavior* of unavailable or difficult-to-access dependencies, as depicted in Figure 1-1. This is done by modeling and deploying a so-called "virtual asset" that emulates those parts of the dependency's behavior that are required to execute the desired test cases. Service virtualization is different from (and complementary to) other types of virtualization by focusing purely on behavior simulation rather than simulating entire systems.

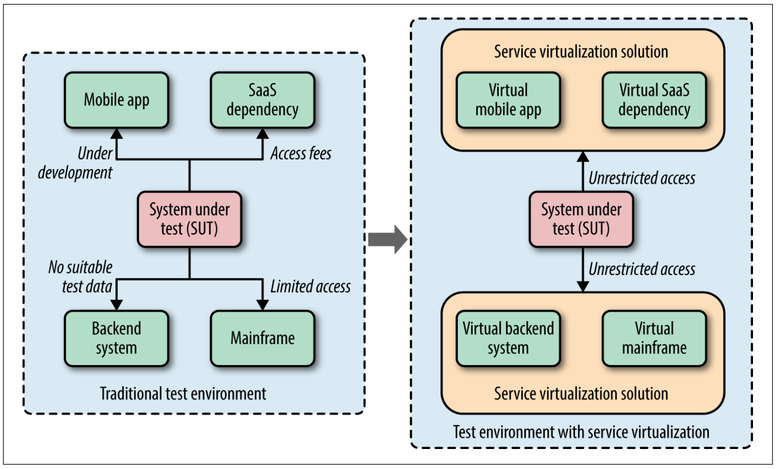

*Figure 1-1. Removing dependency access restrictions with service virtualization*

## Bottlenecks in the Software Development Life Cycle

In the previous section, we explored some of the reasons for the introduction of service virtualization. At the heart of these reasons are three main bottlenecks addressed by the 2015 voke Market Snapshot on Service Virtualization:[2]

*Work is delayed while waiting for dependencies*
: No less than 81 percent of the participants experienced development delays due to access restrictions with regards to environment dependencies. For testing activities, the percentage is even higher: 84 percent of the participants experienced delays in testing activities for the same reason.

*Access to required dependencies is restricted*
: On average, participants said they required 52 dependencies throughout the software development process (up to, but not including deployment into the production environment). However, only 23 of these dependencies can be accessed without restrictions.

*Challenges in accessing third-party dependencies*
: Seventy-nine percent of participants reported that they experienced restrictions accessing third-party dependencies. These restrictions include access fees, time limits, and limited availability of suitable test data.

## How Does Service Virtualization Compare to Stubbing and Mocking?

Development teams have been simulating dependencies required for executing different types of tests (such as unit tests and integration tests) for a long time. Two techniques that have been, and still are, widely used are *stubbing* and *mocking*:

- Stubs are objects that replace a dependency by providing predefined responses to input delivered during tests. Stub behavior

2 This market research was carried out in 2014 using 505 participants from a wide range of organizations, market segments, geographies, and roles.

therefore is predetermined and fixed, making stubs suitable for state verification during test execution.

- Mocks are similar to stubs, with the difference being that the behavior of mocks is defined during test initialization. This means that two instances of the same mock can behave differently, depending on their initialization, which makes them suitable for behavior verification during test execution.

Table 1-1 summarizes the differences between these techniques and service virtualization:

*Table 1-1. A comparison between stubbing/mocking and service virtualization*

| Stubbing and mocking | Service virtualization |
|---|---|
| Used mostly to support unit and unit integration testing | Used primarily to support system, acceptance, and performance testing, although it can be used to support unit and unit integration testing just as well |
| Created and used mostly by developers | Can be created by any authorized individual and then shared and used within the team or across teams |
| Used mostly within the confines of a single team or project | Can be used at an individual team level, across teams within the same project as well as throughout the organization; for example, through a dedicated service virtualization Center of Excellence |
| Does not scale well to support larger projects and test scenarios | Scalable and reliable solution, able to support large projects and test scenarios |
| No support (or need) for data-driven stubbing or mocking | Supports creating flexible, data-driven virtual assets |
| No support (or need) for wide range of message and transport protocols | Supports a wide range of message and transport protocols |

From this comparison, you could conclude that service virtualization takes the support for testing that is provided by stubbing and mocking to the enterprise level.

Now that we have seen what the concept of service virtualization entails, in the next chapter, we'll take a look at how we can make it an integral part of the software development life cycle.

CHAPTER 2

# Service Virtualization Implementation

In this chapter, we take a look at some of the questions you should ask yourself when considering integrating service virtualization into your Continuous Delivery pipeline.

## Selecting a Service Virtualization Approach

The first question that you should ask when considering service virtualization is whether it is necessary in the first place. Service virtualization implementation is not a quick fix, as you will see in the next section. Although the ease of use and the speed of creation of virtual assets is one of the drivers of service virtualization success, its implementation requires proper planning and, depending on the implementation scale, a significant investment of time and effort. This applies especially to organization-wide service virtualization projects. Of course, it is very well possible to begin the implementation process on a small scale to test the waters.

One of the alternatives to service virtualization could be to develop, deploy, use, and maintain your own set of stubs. Be sure to take into consideration the additional costs associated with software maintenance. These costs, including corrective maintenance costs (costs associated with fixing defects) and enhancements (costs associated with continuing innovations), often exceed the initial development and implementation costs significantly and you should not overlook them.

After you establish that service virtualization is the way forward, it is advised that you take some time to identify the bottlenecks in your test environment that are most painful when it comes to delays in development and testing progress. Here are some questions that could be asked:

- Which dependencies are responsible for the delay that your team is experiencing?
- What gains are to be won with service virtualization implementation?
- Do these gains really solve the problem at hand?
- Are there any other quick wins that can either speed up the development process or at least result in management buy-in (preferably both!)?

After you have decided on what bottlenecks to attack first, it is time to consider the various options and tools available on the market. There are a lot of options available nowadays, and one of the most important choices you need to make is between open source solutions (such as Hoverfly from SpectoLabs and WireMock, which is developed and maintained by Tom Akehurst) and commercially licensed service virtualization engines (such as HPE Service Virtualization and Parasoft Virtualize).

Table 2-1 presents some considerations that you need to take into account when making this decision.

*Table 2-1. Considerations related to the choice between open source and commercial service virtualization solutions*

| Open source | Commercial |
|---|---|
| No license fees | License fees (often per "hit" = request/response exchange) |
| No vendor lock in | Vendor lock in (no migration strategies from one commercial vendor to another exist [yet]) |
| Limited functionality | Wide range of options with regard to message and transport protocols, data-driven virtual assets, deployment, and management options) |
| Limited support (often on a best effort basis) | Professional support, on site as well as via email/telephone. |
| Limited scalability | Highly scalable for use throughout the organization |

## Fitting Service Virtualization into Your Software Development Life Cycle

After you have decided on the services that you want to virtualize and the tool you'll use to perform the task, it is time to begin the development of your virtual assets. Similar to the creation of automated test scripts, *you should treat developing and implementing virtual assets as software development.*

This entails the following:

- Development of virtual assets should be made a *specific and visible task* in your overall software development process. In case of Agile/Scrum software development, for example, you should make virtual asset development and maintenance visible as tasks on the Scrum board, assign an owner, and ensure that it is accepted at the end of a sprint. We discuss this further in Chapter 5.
- You should adequately plan for tasks related to the development, implementation, and maintenance of virtual assets, both in time and with an eye on what resources are needed.[1]
- You should treat virtual assets like any other software artifact delivered by a development team. This includes bringing those assets under version control and testing them to validate that they meet expectations.

Taking care of these tasks facilitates the successful embedding of service virtualization into the software development process and ultimately, in the organization as a whole.

It must be noted, though, that a successful service virtualization cannot exist without a high-quality set of tests that exercise the application under test, and, by reference, the virtual assets. This set of tests should not only require basic or default virtual asset behavior, but also cover any edge cases that have been modeled in the virtual asset. If the test set fails to do this, and the virtual asset exerts only

1 Note that one of the success factors of service virtualization is the fact that virtual asset development typically takes a fraction of the time it takes to develop the application itself. As such, adequate planning of virtual asset development does not need to imply planning as rigorously as is done with regular software development tasks.

straightforward and simple behavior, the added value of service virtualization is reduced significantly.

## Benefits of Applying Service Virtualization to Your Software Development Life Cycle

The use of service virtualization to simulate the behavior of critical yet hard-to-access application components and dependencies in your software development life cycle has a number of significant benefits:

- It allows for earlier testing.
- It allows for continuous testing.
- It enables development teams to increase their test coverage.

In the following sections, we'll take a closer look at each of these benefits. Note that there are other benefits as well, but we will not be covering those in this chapter. Here are some examples of these additional benefits:

- The ability to closely replicate dependency performance characteristics for performance testing purposes
- The ability to replicate and fix defects faster
- The ability to prepare functional testing scenarios earlier and execute them faster

### Earlier Testing ("Shift Left")

When you have virtual assets that simulate dependency behavior at your disposal at the beginning of your software development activities, no more time is wasted waiting until every dependency is in place before you can begin integration and end-to-end testing. The ability to test earlier means that potential defects are uncovered earlier in the development process, when they are relatively easy, fast, and less expensive to fix.

The ability to test earlier in the software development life cycle is commonly known as the *shift left* of testing. Figure 2-1 illustrates this concept.

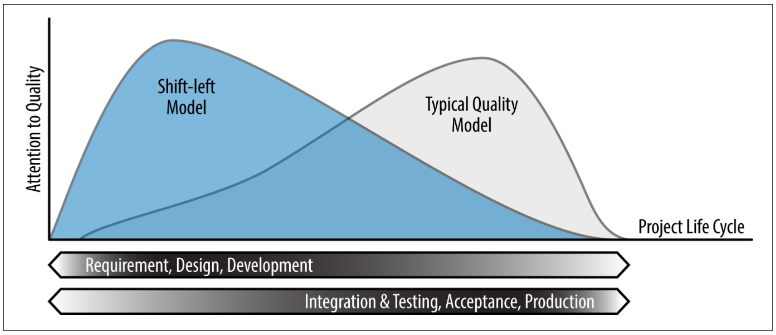
Attention to Quality
Shift-left Model
Typical Quality Model
Project Life Cycle
Requirement, Design, Development
Integration & Testing, Acceptance, Production

*Figure 2-1. Shift left: test earlier in the process to speed up development and find and fix defects faster*

## Continuous Testing

When the IT landscape of your organization is one with a lot of applications running on many different platforms, setting up and maintaining test environments requires a significant amount of time and effort. Chapter 1 demonstrated how service virtualization is—or at least can be—a suitable approach to simplify test environment management and make it more flexible.

With the burden of provisioning and configuring (possibly many) test environments out of the way, and with no more need to populate and repopulate these test environments with the appropriate test data before every test run, you can run tests against the exact same dependency in the exact same state over and over again.

An immediate benefit is that this improves the stability of a software system and enhances the trust stakeholders place in it. It's one thing to see that a specific test or test set passes once; it's another thing altogether to know that that system passed the same set of tests many times.

Another advantage is, in case a defect does occur, being able to recreate the state of the test environment with a single click of a button makes reproduction and root cause analysis of the defect much easier.

The ability to run tests over and over again, on demand, is also known as *continuous testing*.

## Increase in Test Coverage

With full control over the behavior exerted by dependencies, software development teams can set up and execute specific test scenarios with far more ease compared to having to deal with "real" dependencies. For example, service virtualization makes it easier to perform the following:

*Negative test scenarios*
: Not only do you want to ensure that your system handles the "happy flow" scenarios correctly, knowing that it can handle incorrectly formatted messages, message timeouts and other exceptions is of vital importance, as well. These scenarios are notoriously difficult to simulate in real test environments, but with service virtualization, it's just another scenario that you can model in your virtual asset.

*Edge case scenarios*
: To ensure that your system can handle all input values that are within range, you will need to cover all edge cases, as well (because those are often the scenarios that cause trouble in a system). Finding or creating suitable edge cases in a real environment can be a laborious task, but with full control over the test dataset contained within a virtual asset, simulating edge case behavior isn't any different from simulating common, middle-of-the-road cases.

*Other hard to set up scenarios*
: An example of this scenario type is the case in which the system you are testing expects messages to arrive in a specific order. For this scenario, you do want to verify the error handling capabilities in case those messages do not arrive in the prescribed order. Again, this is much easier to accomplish when you have full control over the behavior exerted by your virtual asset.

In this chapter, we covered how you can make service virtualization an integral part of the software development life cycle and what benefits you can gain by doing so. In the next chapter, we'll take a closer look at these benefits in light of the bimodal IT practice model.

# CHAPTER 3
# Service Virtualization in a Bimodal IT World

In 2014, Gartner published a new practice model for organizations that deliver software: *bimodal IT*. In short, an organization that follows the bimodal IT practice adopts two separate styles (modes) of software development and delivery:

- Mode 1—also known as reliability mode—is focused on predictability. This involves development projects that are related to core system maintenance, stability, and efficiency. These projects generally require little business involvement and are often organized in a traditional, sequential manner.
- Mode 2—or agility mode—focuses on innovation and differentiation. This involves finding solutions for new problems and exploration of areas of uncertainty. These projects generally require a high level of business involvement and are often organized in an incremental, short-cycled, Agile method of working.

Figure 3-1 shows an overview of what bimodal IT entails.

Mode 1 is mostly targeting systems of record—systems that store information, often both current and historical, and that are used to keep the daily business running. Mode 2, on the other hand, is mainly focusing on systems of innovation—systems that are meant to provide an organization with an edge over their competition.

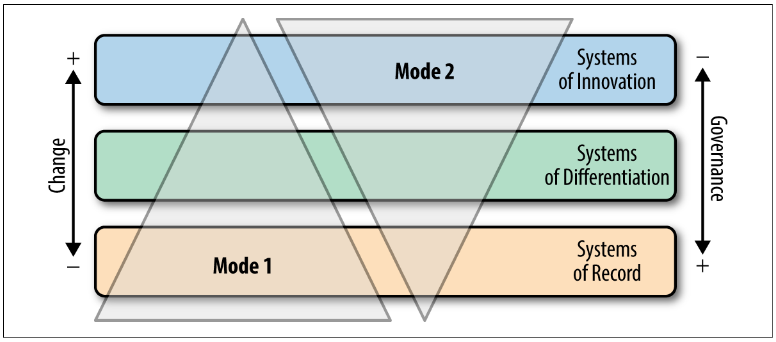

*Figure 3-1. A bimodal IT approach*

In this chapter, we take a look at the role service virtualization can play in software development projects in either mode, and how organizations can benefit from an enterprise-level service virtualization approach spanning both modes.

## Reliability Mode and Service Virtualization

Even though a growing number of software development projects are organized in a short-cycled and incremental manner in order to be able to react to the market's demand for speed, projects that follow the traditional ("waterfall") approach still exist and will probably exist for some time to come. For these projects, too, service virtualization has a number of potentially very valuable advantages.

Reliability mode projects often involve maintenance, upgrading or phasing out of large, monolithic systems, such as mainframes or Enterprise Resource Planning (ERP) systems. Creating and provisioning test environments for such systems can be a very time-consuming and expensive process. Modern service virtualization systems that offer the ability to simulate the behavior of such systems can bring about significant cuts in expenses for test environment setup and maintenance.

Also, after virtual assets that simulate a mainframe or ERP system have been created, development teams do not need to claim time from the scarce resources that possess detailed knowledge of these systems anymore (or at least not as often). Any person who has knowledge of how the service virtualization solution is set up can add or adapt behavior (or test data) required for specific test cases.

This removes another potential bottleneck from the development and testing process.

Another aspect of service virtualization that is especially beneficial to reliability mode projects is the ability to simulate performance characteristics of dependencies. Take for example the situation in which an organization wants to periodically monitor end-to-end performance in a production-like environment to ensure that changes to specific systems do not have a negative impact on overall performance. If these end-to-end tests involve applications or systems that are expensive to replicate in a test environment (such as the aforementioned mainframe or ERP systems), having a virtualized replica of these systems, in terms of both functionality and performance, in place can make end-to-end performance monitoring much easier, or even possible at all.

## Agility Mode and Service Virtualization

Whereas the benefits of service virtualization in reliability mode focus on cost savings and the virtualization of complex and difficult-to-duplicate systems, the benefits for development projects in agility mode are centered around enabling and improving speed of delivery.

When multiple development teams are working on separate parts of an application that must interact with one another eventually, waiting until all of the development teams have finished developing before integration testing can take place is a time-consuming process. This loss of time grows linearly with every rework iteration required in case defects are found. With service virtualization, development teams can expose the behavior of the artifacts they are developing before the actual component is finished. This enables other teams to test their own work against the component simulation faster, thereby greatly increasing the speed with which they can find defects and subsequently resolve them.

With service virtualization, the impediment of having to share test environments with other teams can be made a thing of the past, as well. Especially when service virtualization is combined with containerization techniques as described in Chapter 4, provisioning multiple instances of the same virtual asset is a matter of minutes. In this way, all teams can create and use their own test environments, tailored to their specific needs, without running the risk of test environment or test data interference.

Finally, because service virtualization allows for quick provisioning of test environment instances, it is an enabler for continuous testing, which is another requirement for development teams wanting to speed up and increase the agility of their software delivery process. For more information, see Chapter 2.

## Bridging the Gap

As described in the previous sections, service virtualization provides significant benefits to software development projects in either mode of the bimodal IT practice. This does not imply, however, that there is no need for an integrated service virtualization approach that spans development projects in either mode. Having organization-wide guidelines and best practices, such as virtual asset design standards, with regard to service virtualization in use provides maximum efficiency. Here are a couple of ways it can do this:

- Virtual assets can be shared and reused across development teams and projects, with each team having the opportunity to load its own datasets and performance characteristics.
- The number of service virtualization engine licenses required to serve all development teams with the virtual assets they need (in case of a commercial service virtualization solution) is minimized.

Another software development and delivery practice that is seeing a lot of followers is Continuous Delivery. In the next chapter, we demonstrate how service virtualization and Continuous Delivery go hand in hand when organizations want to deliver quality software at the speed demanded by increasingly competitive markets.

CHAPTER 4

# Service Virtualization and Continuous Delivery

This chapter provides an overview of the role that service virtualization can play in an organization that uses Continuous Delivery (CD) to maximize their speed of application development and release. We'll take a look at how both concepts fit together, and what additional benefits are gained from containerizing your virtualized test environments.

## The Role of Service Virtualization in the CD Process

Organizations that are embracing CD as a means of bringing software to production in a fast and flexible manner can't do without the ability to test their applications in a continuous and automated manner. This implies that suitable (with regard to configuration and test data, for example) test environments should be available on demand, something that is proving to be very difficult to do without service virtualization.

Although development teams usually can get by with dependency mocking in the early stages—most notably when writing and executing unit tests—it is becoming increasingly difficult to manage real test environments in the later stages (for integration and end-to-end testing) in such a way that continuous testing can be facilitated. Two of the most important problems related to this are synchronization of test data across dependencies and lack of availability of all dependencies required at the same time.

Building and using simple stubs might work when the technology and behavior to be simulated is relatively straightforward. But modern heterogeneous environments and composite applications comprising a lot of different types of components and dependencies (web services, databases, mainframes, ERP systems, SaaS solutions) require a more sophisticated, enterprise-level solution. This is exactly where service virtualization comes into play.

We need to give you one warning, though: even though service virtualization has potentially many benefits to your development and testing activities, you shouldn't rely solely on it and forget about "the real world." There are some very good reasons for methodically testing against real dependencies:

- While setting up virtual assets, you might choose not to model certain dependency characteristics for the purpose of speed or flexibility. This applies especially to nonfunctional aspects such as security and performance. To be sure that your system under test is able to cope with these aspects, as well, test it against real dependencies.
- Over time, your virtual asset definition might become out of sync with the dependency being simulated. Although it's no substitute for proper communication and version control, testing against the real dependency every now and then is a good way of detecting these changes.

A typical CD process takes care of the above by having code tested against an increasing number of real implementations as the code progresses through the pipeline. In the early stages, tests are run mainly against simulations as a way to balance cost of dependency management and speed. Here, the majority of defects will be, or at least should be, caught. As the code moves closer toward the production environment, virtual assets are traded for real dependencies, as depicted in Figure 4-1.

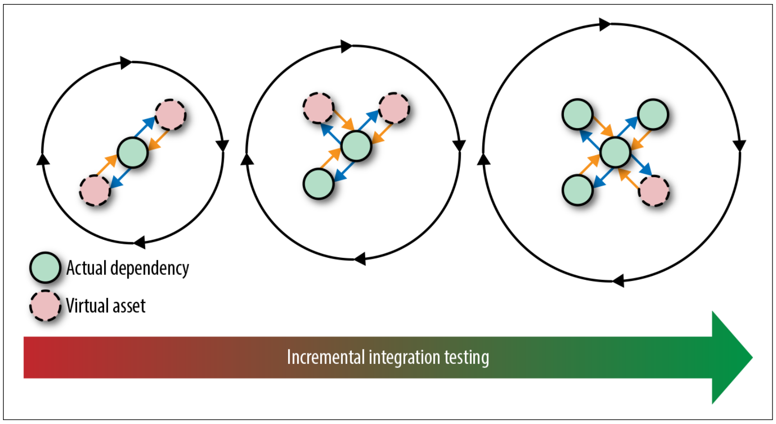

*Figure 4-1. Trading virtual assets for actual dependencies as code moves through the Continuous Delivery pipeline*

## Containerizing Virtual Test Environments

Using service virtualization as an enabler for CD brings many benefits of its own. However, when we treat the virtual environments as artifacts in that CD process, just like the system under test itself, and combine this approach with the power of cloud computing, we really take service virtualization to the next level.

A sample CD cycle, powered by containerized service virtualization, could follow a path that looks as follows:

1. A developer commits his code changes to a version control system such as Git or Subversion.
2. The build server, for example Jenkins or Atlassian Bamboo, triggers a new build and run unit tests to ensure code quality on the unit level.
3. After unit tests pass, the system under test is deployed on a test environment that is created dynamically on a public or private cloud server.

   Simultaneously, a virtual test environment is created and provisioned on its own public or private cloud server. Test environment configuration is set so that the desired behavior, test data sets and performance characteristics (for example) are enabled.

4. Tests (system, integration, or end-to-end) are run against the system under test, which in turn communicates with the virtual assets in the simulated test environment.
5. After these tests pass, the system under test is deployed safely into a production environment. Meanwhile, the cloud test environments are deprovisioned and removed, ready to be re-created in the next cycle.

Figure 4-2 offers a graphic representation of this process.

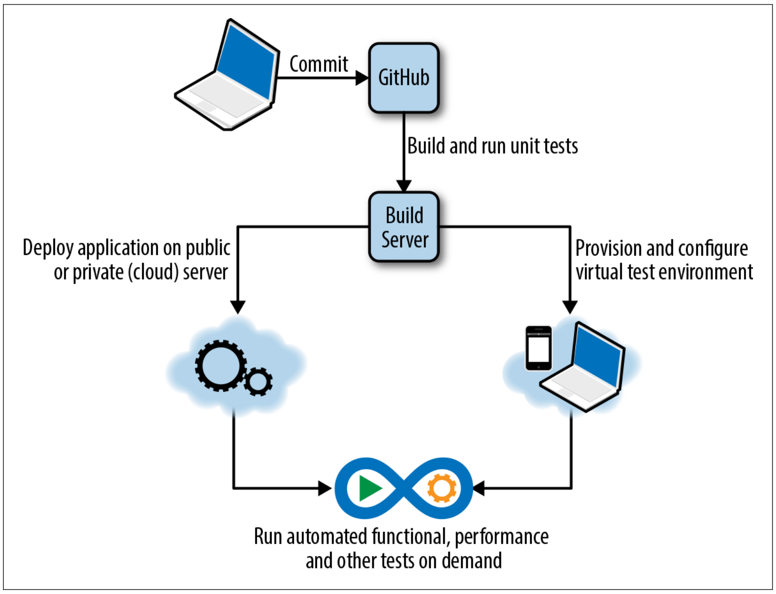
Commit
GitHub
Build and run unit tests
Build Server
Deploy application on public or private (cloud) server
Provision and configure virtual test environment
Run automated functional, performance and other tests on demand

*Figure 4-2. Containerized service virtualization as an enabler for CD*

In this chapter, we looked at how you can combine CD and service virtualization to deliver quality software at the speed that today's competitive markets demand. The next chapter discusses the role service virtualization plays in and the associated benefits for a number of other IT trends.

CHAPTER 5

# The Role of Service Virtualization in Current IT Trends

In this final chapter, we take a look at how service virtualization plays (or can play) a role in some of the major IT trends from recent years, such as Agile and lean software development and the Internet of Things. We'll explore realistic scenarios in which service virtualization has a positive impact on software development and testing efforts within each of these trends.

## Agile Software Development

The practice of Agile software development, especially its popular Scrum variant, involves cross-functional teams that incrementally deliver working software every couple of weeks. In each iteration, or sprint, software is developed, tested, and presented to the stakeholders, which might lead to adjustment of the goals for the next sprint(s) or even for the remainder of the development trajectory.

Being able to deliver working software iteratively requires of the development team that it be able to rapidly and continuously assess the quality of the product, often through means of automated testing. This is impossible without having the required dependencies and test environments in place and correctly configured throughout each sprint.

If the team must wait until the end of every sprint to perform the necessary testing because dependencies have not been delivered on time or test environments are otherwise unavailable, the risk of sprint goals not being met and the project degrading to "iterative waterfall" increase significantly.

With appropriate service virtualization in place, however, Agile development teams are able to begin testing as soon as the first deliverable becomes available, and they can continue monitoring quality throughout the sprint. This greatly increases the odds of delivering quality software providing the desired business value at the end of each iteration.

## The Internet of Things

The *Internet of Things* (IoT) is the network of devices, vehicles, buildings, and other objects featuring network connectivity, which makes it possible for them to communicate with one another in order to collect and exchange data.

You can consider these network-enabled objects as a specific form of dependency when developing and testing software that communicates with these devices. For example, when you're build a smartphone app with which users can view the inventory in your IoT-enabled fridge, that fridge is a dependency in your test environment just like any other dependency we have seen.

Here are two IoT-specific challenges when it comes to test environment management and the way service virtualization can provide a solution:

- The cost of setting up and maintaining a physical test environment can become a serious burden when your software communicates with a large number of IoT-enabled devices (such as different types of cars). Creating a virtual asset that simulates the behavior exerted by these devices can quickly become a very cost-effective alternative.
- A challenge specific to testing in the IoT is the ability to test under different performance and network characteristics (for example, with high latency or poor network reception). It can be difficult to simulate these characteristics in a test lab with real devices. Implementing a service virtualization solution that can be configured to simulate different types of performance and network characteristics gives teams the ability to test their soft-

ware under all types of circumstances instead of under only the specific conditions in their test lab.

## The API Economy

The term *API economy* refers to the trend of organizations exposing (part of) the business logic in their software through APIs with which the outside world can interact. For many of the world's largest software organizations, such as Google, Amazon, and Facebook, exposing their data and business logic through APIs has become a proven strategy for revenue increase.

For software development teams looking to build applications that connect to and communicate through these APIs, service virtualization can be a useful strategy to mitigate some of the risks associated with tests incorporating components outside of their circle of control:

- Setting up the required test data for specific test cases can be a laborious task (if it's possible at all) when dealing with external APIs. By creating virtual assets that simulate the behavior of these APIs, control is regained and test coverage and test cycle speed can be increased.
- Even though a large part of the APIs available are free to use—at least to some extent—organizations might charge access fees for specific APIs or only provide a fixed number of requests/responses for free. This is where the "economy" part of the "API economy" comes into play. If continuous (performance) testing is part of your overall testing strategy, these access fees can become a serious expense. In that case, service virtualization can be a flexible and cost-saving alternative.

## Lean Software Development

The practice of lean software development[1] focuses on the elimination of waste in software development, where 'waste' is defined as

1 The principal book on lean software development is *Lean Software Development: An Agile Toolkit*, by Mary Poppendieck and Tom Poppendieck, ISBN 978-0-321-15078-3.

work that does not value to a product or service. Lean software development is summarized by seven principles:

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

Service virtualization can be a significant contributor to most of these principles, and therefore to the concept of lean software development as a whole. For example:

*Eliminate waste*
: A prime example of waste in software development is waiting time. By replacing dependencies that are not yet developed or that software development teams need to schedule access time for with virtual assets that are readily available, waiting time is eliminated.

*Amplify learning*
: In lean software development, learning is amplified by the use of short development and testing cycles that allow development teams to quickly implement and evaluate possible solutions to the problem of software design. This relates to the concepts of continuous testing and shift left, as introduced in Chapter 2.

*Deliver as fast as possible*
: To ensure that software that is delivered fast does still meet the predefined quality threshold, the ability to test early and test often is indispensable to software development teams. Again, refer to Chapter 2 for more information on how service virtualization is an enabler for this so-called “quality at speed.”

## What Will the Future Hold?

From this chapter and the previous chapters, we can conclude that service virtualization is a technique that provides significant benefits in terms of time, effort, and money saved to software development processes of all kinds.

As of this writing (2016), the service virtualization market is still maturing, with commercial service virtualization solutions (such as HPE Service Virtualization and Parasoft Virtualize) becoming ever more powerful and rich in features. We're also witnessing a growth in open source solutions (such as Hoverfly and WireMock), often geared toward a specific aspect or type of service virtualization.

Of course, the future cannot be predicted with any kind of certainty, but when we take the above into account, while also looking at IT market predictions from established researchers such as Gartner and Forrester, it is inevitable that service virtualization will remain a practice to watch and explore for the years to come.

# APPENDIX A
# Service Virtualization Case Studies

This appendix contains a number of case studies that illustrate the benefits that service virtualization brings to the software development life cycle.

## Retail

A global, multichannel retailer turned to service virtualization to speed up its time to market by shifting-left its testing efforts. For this organization, faster time to market implied rapid development, testing, and deployment of applications serving both its internal (such as the HR department) and external customers (served through the website and stores).

The organization applied service virtualization to improve its testing quality and its Continuous Delivery and integration capabilities. Critical yet hard-to-access third-party systems have been replaced by virtual assets, which enabled the organization to test faster, more often, and earlier in the software development life cycle. This in turn led to a transition from a traditional waterfall to an Agile/DevOps development approach.

Removing the dependency on third-party application by means of service virtualization also helps this organization improve its performance testing efforts by enabling development teams to execute performance tests earlier and more often than was previously possible.

## Finance

A top-four bank was looking for a solution that allowed it to adapt to increasingly fast-paced and complex application development cycles without compromising application quality. Initially, it was building simulators to replace third-party services required for end-to-end and performance testing as well as external mainframes that had limited availability (they could be accessed only during specific time slots). Building these simulators took on average around 50 hours, effectively extending the bank's testing cycle by two weeks.

By replacing the ad hoc simulation development with service virtualization on an enterprise-wide scale, the time needed to create a simulator has been reduced to a mere four hours. Also, performance test cycle duration has been reduced by on average two weeks, allowing the test team to be more flexible and better support Agile development.

## Advertising

A global-operating marketing and advertisement company needed to update their web services and applications weekly; in the past, the company did this only three or four times a year. However, testing efforts proved to be time consuming, costly, and complex due to the number and variety of devices that the services and applications needed to support.

By implementing service virtualization to replace the wide range of devices and configurations required for adequate testing with manageable and configurable virtual assets, the company was able to take back control over its testing efforts. It also enabled the company not only to simulate normal device behavior, but to test the functionality of its services when facing performance issues, as well.

Additionally, the time needed to set up and provision a testing environment has been reduced from two weeks to two days, with significant cost reduction (in the range of several hundreds of thousands of dollars) as an added bonus.

## Further Reading

*Service Virtualization for Dummies* by Marcia Kaufman and Judith Hurwitz, John Wiley & Sons, Inc., 2013. ISBN 978-1-118-50127-6

*Service Virtualization: Reality is Overrated* by John Michelsen and Jason English, Apress, 2012. ISBN 978-1-430-24671-8

## About the Author

**Bas Dijkstra** is a test automation and service virtualization consultant with more than 10 years of experience helping his clients bring their software testing efforts to the next level via smart application of tools. He has successfully designed and implemented test automation and service virtualization solutions for clients in a multitude of sectors. Bas is also an experienced teacher, writer, and speaker on several topics related to test automation and service virtualization. He lives in the Netherlands with his wife and two sons.