# Service Virtualization FAQ

– getting access to complete end-to-end test environments on demand –

## Abstract

Microsoft and Parasoft are building an alliance to facilitate Cloud migrations for organizations with a complex application landscape. In particular for the migration of non-production environments (such as development, integration, acceptance and performance testing activities).

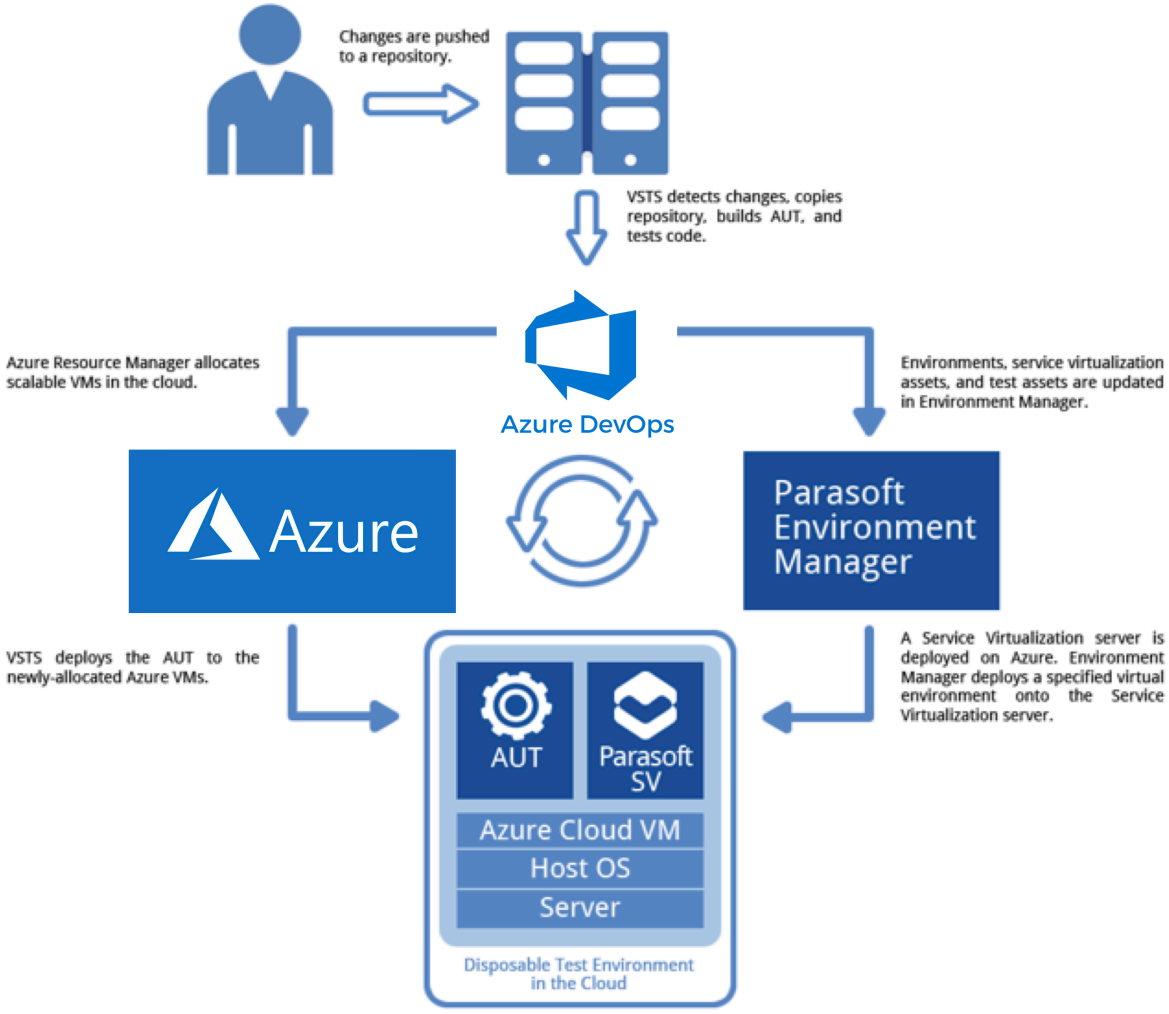
The costs and efforts for migration are a well-known bottleneck for Cloud migration. This FAQ will introduce the high-level concept, as well as answer the frequently asked questions.

# Table of Contents

## Concept overview

Modern software applications are highly interconnected with other systems (either internal or external services). When migrating to a system from an on-premise situation to an Azure / Cloud based solution (IaaS, PaaS), the migration of the connected service is a huge bottleneck. To remove those dependencies, Parasoft provides Service Virtualization to offer light and easy to deploy simulations of those dependencies.

De diagram below shows the solution, where AUT indicates the Application Under Test (the application that needs to be migrated) and Parasoft SV is handling all the dependencies that are needed for the AUT to function correctly.




As shown in this diagram, Parasoft Service Virtualization is completely Azure based and provides a flexible solution including test environment management.

## 1. What is Service Virtualization?

In a nutshell, service virtualization provides teams with easy access to the constrained components that impede development and testing. This usually manifests as environmental constraints, in which components that are technically out of scope for testing are required in order to enable full end-to-end functionality.

With service virtualization, you can remove these constraints by simulating those downstream dependencies and swapping out the real functionality with emulated behavior.

When done correctly, the system behaves just as if the actual component was available. Thus, you can eliminate scheduling constraints by providing ubiquitous access to an accurate emulated test environment. And you can eliminate process bottlenecks by providing rapid access to evolving, unavailable, or otherwise difficult-to-access dependent systems.

As stated by Wikipedia's service virtualization entry, these dependent systems might be:

- Not yet completed
- Still evolving
- Controlled by a third-party or partner
- Available for testing only in limited capacity or at inconvenient times
- Difficult to provision or configure in a test environment
- Needed for simultaneous access by different teams with varied test data setup and other requirements
- Restricted or costly to use for load and performance testing

Wikipedia's entry continues to describe this well:

> *"Rather than virtualizing entire systems, it virtualizes only specific slices of dependent behavior critical to the execution of development and testing tasks. This provides just enough application logic so that the developers or testers get what they need without having to wait for the actual service to be completed and/or readily available.*
>
> *For instance, instead of virtualizing an entire database (and performing all associated test data management as well as setting up the database for every test session), you monitor how the application interacts with the database, then you emulate the related database behavior (the SQL queries that are passed to the database, the corresponding result sets that are returned, and so forth)."*

## 2. Why is service virtualization important?

To achieve quality at speed, it's essential to have unrestrained access to a trustworthy and realistic test environment. It is important to recognize that a complete test environment includes the application under test (AUT) and all of its dependent components (e.g. APIs, 3rd-party services, databases, applications, and other endpoints).

Service virtualization enables DevTest teams to get access to a complete test environment, including all critical dependent system components, as well as alter the behavior of those dependent components in ways that would be impossible with a staged test environment — enabling you to test earlier, faster, and more completely. It also allows you to isolate different layers of the application for debugging and performance testing, but we're not going to get into that as much today.

## 3. How does it help in getting access to a complete test environment?

With today's fast-paced iterative development cycles, DevTest teams need early access to a complete test environment in order to:

- Validate each user story's functionality as soon as it is completed
- Validate each user story's impact as soon as it is completed
- Perform more comprehensive testing earlier in the process
- Complete their own DevTest tasks even if another team is implementing or evolving a dependent component in parallel with the current iteration

Service virtualization can provide access to any dependent component that is missing or constrained in your test environment: 3rd-party services, APIs, databases, mainframes, ESBs, and other components that communicate using common messaging protocols. Prime candidates for service virtualization include dependent components that are both:

- Moderately (or more) difficult to access for testing—for example, due to scheduling conflicts, access fees, licensing, geo-political boundaries, etc.
- Moderately (or more) complex to configure for testing

For example, an internal service might be readily accessible from a staged test environment and simple to configure. On the other hand, a complex message queue is probably more difficult to stand-up in a staged test environment and considerably more challenging to configure for test. At the extreme end of the spectrum, a mainframe or ERP system will have multiple constraints associated with DevTest access as well as distinct limitations on your ability to configure it for test. Leveraging service virtualization ensures that a test environment is accessible on demand. It eliminates the access constraints and reduces the overhead associated with repeated configuration.

### 4. Can you alter the behavior of dependent components?

Service virtualization also gives you control over the behavior of the dependent components. It is very difficult to alter the configuration of the network or hardware associated with each dependent component of the AUT. It's also quite common to face staged test environments that exhibit slower performance than you'd encounter in production.

Using service virtualization, you have greater control over how dependencies respond. This gives you on-demand access to a much broader range of dependency behaviors (just like a flight simulator). As a result, you can assess the risk of a release candidate faster and more accurately.

For example, you can simulate different dependency behavior to:

Check how your AUT responds to performance variations in dependencies. Can users complete core transactions even when one dependency experiences high latency? Do low-latency scenarios expose concurrency issues?

Isolate the component under test to understand if unexpected behavior stems from problems with dependencies or from your AUT

Set the complete test environment into different states and validate your AUT's security and resiliency in those contexts

### 5. Can you alter the data of dependent components?

Virtual services do not need to always respond with the actual data in the real system. In fact, there are many benefits to providing unexpected data from your virtual services. Virtual services are separated from their data sources, which allows much greater flexibility in generating response data that suits different teams needs, such as:

Development teams that want to guard against malformed responses or negative behavior in their application can generate service responses that provide negative behavior.

Testing teams that want to validate how non-nominal responses are handled by the service can return illegal characters in the response.

Performance teams that want to understand the impact of large payload responses can provide larger-than-normal responses from dependent components.

By simulating the different service data in these types of situations, you can gain much more flexibility with your testing.

## Service Virtualization and cloud migrations

### 6. How can Service Virtualization facilitate in migrating to the cloud?

Migrating to the cloud might be jeopardized by several aspects. SV allows to reduce risk by enabling teams to test earlier, faster & more completely. This means that teams can perform testing scenarios without delays or problems with access to "not-ready-yet" elements.

SV can be used to confirm correctness of migration or the configuration process after migration. It allows to confirm proper behavior of migrated systems with different boundary conditions or simulation of latency.

### 7. How can Service Virtualization increase the success rate of the different projects going into the cloud?

There are several reasons why Service Virtualization will help in increasing the success rate of the different projects going into the cloud, such as:

- OpEx reduction by cutting wait time, configuration time & faster behavior modeling.
- Risk reduction due to QA and performance testers being able to simulate missing or evolving system components to incrementally test applications earlier and more completely.
- Efficiency grows due to faster testing environment creation.

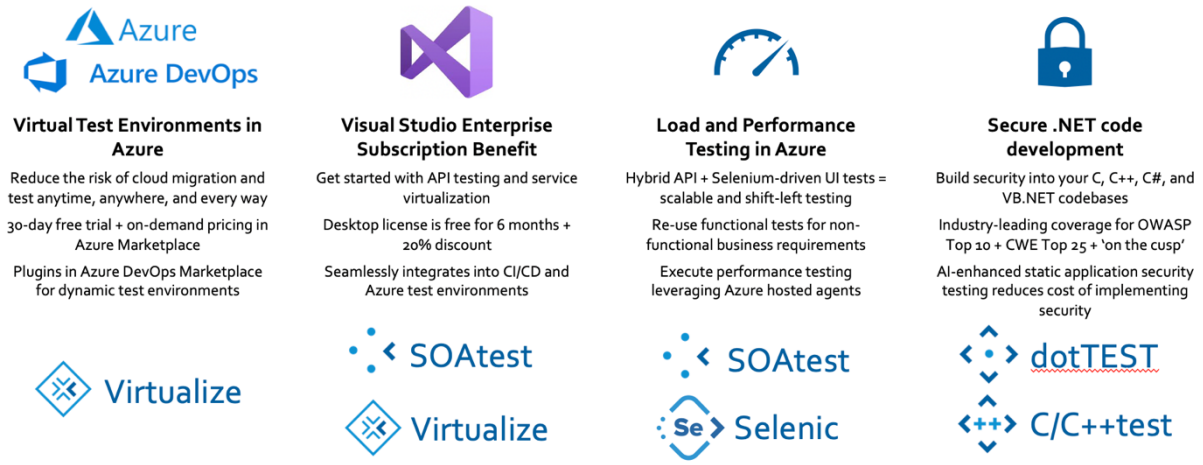
### 8. What other benefits can Service Virtualization bring to cloud migration / deployments?

Service virtualization delivers a simulated test environment that allows you to test earlier, faster, and more completely.

Apart from facilitating in cloud migration and enhancing the success rate of the projects going into the cloud, additional benefits Service Virtualization can bring are:

- Simple and easy configuration and provisioning of environments for different test scenarios.
- Broad, flexible support for protocols, transports, message formats.
- Graphical, environment-driven approach to simulation and provisioning.
- Rapidly prototypes functionality in order to accelerate feedback cycles & reduce time to market.
- Faster and easier bug/malfunction detections ("Shift left" process)

9. Where can I find out more on Parasoft SV for Azure?




* Service Virtualization for Azure DevOps: https://marketplace.visualstudio.com/items?itemName=parasoft.ctp-vsts-extension

* Service Virtualization in Azure: https://azuremarketplace.microsoft.com/en-us/marketplace/apps/parasoft.parasoft-service-virtualization

* Plugins API Testing for Azure DevOps: https://marketplace.visualstudio.com/azuredevops

* Visual Studio Enterprise Subscription for Service Virtualization + API Testing: https://visualstudio.microsoft.com/subscriptions/

* Supporting Both Visual Studio and Visual Studio Code: https://marketplace.visualstudio.com/vscode

* Landing Page: Streamline Your Migration to Microsoft Azure With SV https://alm.parasoft.com/microsoft-service-virtualization

* Technical Guide: Deploying Virtualize to Cloud-based Microsoft Environments Skip to end of metadata https://docs.parasoft.com/display/GUIDES/Deploying+Virtualize+to+Cloud-based+Microsoft+Environments

## What are the components in the Parasoft SV solution?

Parasoft Service Virtualization consists of the following components:

- Virtualize Server: Application (based on Tomcat container) that runs the Virtual Assets
- Virtualize Desktop (Windows, Linux and OS X) for the creation of Virtual Assets
- Data Repository Server (DR): On MongoDB based
- Continuous Testing Platform (CTP): Application (based on Tomcat container) to manage the server instance:
    - Thin client to construct Virtual Assets
    - Thin client for Provisioning of Test Environments
    - Thin client for Test Data Management