PARASOFT®

WHITEPAPER

How to Choose the Right **Service Virtualization Solution**

Key Capabilities to Make Your Organization Successful in Today's Testing Environments

WHAT IS SERVICE VIRTUALIZATION?

Providing an effective way to simulate dependent services that are out of your control for testing, service virtualization is a key enabler to any test automation project. By creating stable and predictable test environments, your test automation will be reliable and accurate. But there are several different approaches and tools available on the market. What should you look for in a service virtualization solution to make sure that you're maximizing your return on investment?

Use this guide to help you identify the key features and capabilities needed for your successful enterprise deployment and adoption of service virtualization. It's more important than ever to choose a solution that's right for you and your organization.

THE SERVICE VIRTUALIZATION TOOL LANDSCAPE

The service virtualization tooling landscape breaks down into two types of service virtualization solutions.

Lightweight Tools

Free or open-source tools (such as Traffic Parrot and Mockito) are great tools for beginners because they help you get started in a very ad hoc way, so you can quickly learn the benefits of service virtualization. These solutions are usually sought out by individual development teams to "try out" service virtualization, brought in for a very specific project or reason.

The downside of the lightweight tools is they struggle to garner full organizational traction because they lack the breadth of capability and ease of use required for less technical users to be successful. Additionally, while these tools are free in the short term, they become more expensive as you start to look into maintenance and customization

Enterprise Tools

More heavyweight tooling is available through vendor-supported tools, designed to support power users that want daily access to create comprehensive virtual services. These solutions are often designed with deployment and team usage in mind.

When an organization wants to implement virtualization as a part of their continuous integration and DevOps pipeline, enterprise solutions integrate tightly through native plugins into their build pipelines. Additionally, these solutions can handle large volumes of traffic while still being performant. Obviously, these solutions are not free, so an organization needs to make the right decision when moving into this level of usage.

How Do You Choose the Solution That's Right for You?

Most organizations won't self-identify into a specific tooling category such as lightweight or enterprise, but rather have specific needs from their solution. The honest answer is that you need the best of both of these worlds, so the best way to choose a service virtualization solution that's right for you is to look at the different features and capabilities that you may require and ensure that your tooling choice has those capabilities. Additionally, you may identify other areas of capability that you may need in the future. Use this guide as a checklist to determine those capabilities that are most important to you both now and in the future.

CORE CAPABILITIES FOR EASE OF USE

- SCRIPTLESS FUNCTIONALITIES
- ABILITY TO RAPIDLY CREATE VIRTUAL SERVICES BEFORE THE REAL SERVICE IS AVAILABLE
- INTELLIGENT RESPONSE CORRELATION
- DATA-DRIVEN RESPONSES
- ABILITY TO REUSE SERVICES
- A CUSTOM EXTENSIBILITY FRAMEWORK
- SUPPORT FOR AUTHENTICATION & SECURITY
- CONFIGURABLE PERFORMANCE ENVIRONMENTS
- SUPPORT FOR CLUSTERING & SCALING

SCRIPTLESS FUNCTIONALITIES

For optimal ease of use, you should be able to build virtual services without having to write any code. A visual tooling system is best for this because quite often the person who requires a virtual service will not be the person who originally developed the service and will not have intimate knowledge of its implementation.

This is important to think about because you will be implementing a lot of business logic into your virtual services, in order to make sure they behave like the real services. Having a visual tooling system allows you to easily approach that task, as well as easily share the implementation with a broader audience because it will be easier to understand. Everybody's coding style is different, and with a purely code-driven implementation it's difficult to look at the virtual service and understand what it is doing. With a scriptless visual system, you can grasp its concept and borrow pieces of the implementation in a much easier way.

A solution that's intuitive and scriptless also enables a large body of nontechnical users to build the right virtual services quickly. By reducing the reliance on a strictly coded implementation, non-developers will be able to help build virtual services, fostering a shift-left approach to virtualization usage and helping it become widely adopted more quickly.

ABILITY TO RAPIDLY CREATE VIRTUAL SERVICES BEFORE THE REAL SERVICE IS AVAILABLE

Virtual services can be leveraged as prototypes, which is a very powerful usage of virtualization. This means creating interfaces for dependent components of your application before they are available. Since you won't be able to use record-and-playback to create these, your service virtualization solution must be able to create virtual services from service definitions such as WSDL, Swagger, Open API (OAS3), schema, example payloads, and so on.

With this capability, you can spin up virtual services as soon as the service definitions have been created, enabling a strategy in which developers can build the virtual services as they are building the real services, so that parallel development can take place and out-of-sync agile development doesn't become a blocker.

Another powerful part of creating the service before it is available is for test-driven development (TDD). Testers can create a simulation of what the service will be, and then start to develop their tests against it so that as the service becomes available, they get a jumpstart on their test automation.

INTELLIGENT RESPONSE CORRELATION

Intelligent response correlation means that a virtual service can respond differently depending on the request coming in. The different types of response correlation must be considered.

Deployment Correlation

Each virtual service should be deployed on individual (or multiple) listeners (HTTP, MQ, JMS, TCP, Kafka, and more). An individual virtual service should only pick up messages that are meant for it, so your virtualization solution should be able to discern different messages coming to different queues/paths.

Message Correlation

As individual messages come in, you may have different operations or resources that should be acted on differently. For example, "add account" and "update account" should go to different pieces of response logic, so your service virtualization solution should be able to analyze the incoming message for patterns and route the messages appropriately.

Data Source Correlation

Once the message has been routed to the appropriate response object, you may want to further slice and dice the message depending on key information in the request. A good example of this would be responding differently to various customer accounts. You may have 100 different accounts that have different types of response bodies, and you don't want to create a response object for every single ID, so your virtualization solution needs to be able to correlate on data in the incoming request and look it up in a data source for response.

DATA-DRIVEN RESPONSES

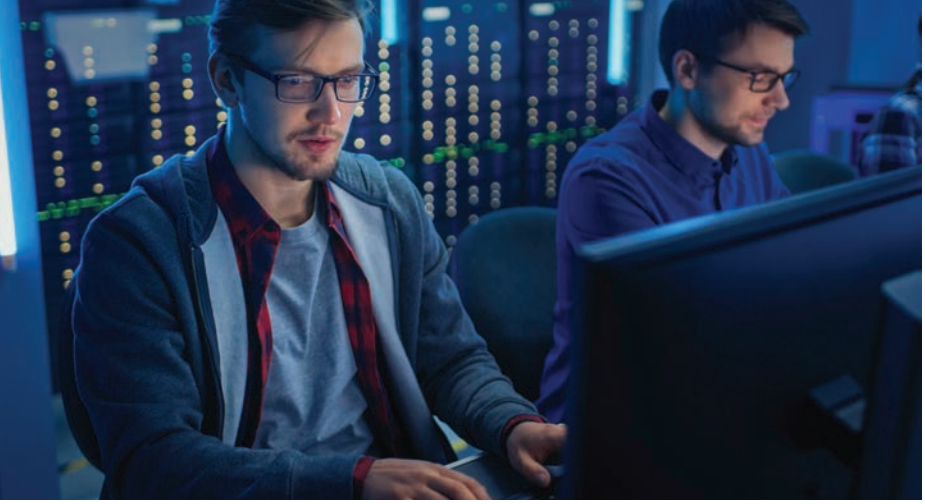
Your virtualization solution should be flexible, so you can create the logic in an abstracted way from the data. Data-driven responses allow you to connect your virtual services to data sources like Excel, CSV, tabular, hierarchical, and even to live databases for real-time data lookups.

Data will be one of the most important pieces of your virtual service because it will be changing all the time. With the ability to parameterize response data in an external file, your virtual services will become more flexible, as well as provide you with the ability to offload some of the response logic into the data source so that it can be handled by the QA or Test Data Management team.

You should also be able to leverage dynamic data sources. An example of this would be if you had transient data provided to your service as a lookup, like an order number, you would want to be able to respond back with the appropriate data even if it didn't exist in your data source. By pulling dynamic data at runtime, as well as storing that data in a stateful way, you will be able to create flexible virtual services that aren't hindered by static data.

ABILITY TO REUSE SERVICES

In order to define core, common, or shared services and reuse those templates and logic in other virtual services, your service virtualization solution should have the ability to repeatedly leverage any virtual service logic you have previously created. This will help the team collaborate because you will be able to come to consensus on what certain key service behaviors should be. Mock that up once, and then use it multiple times.



A CUSTOM EXTENSIBILITY FRAMEWORK

Should you need to write code to accomplish tasks such as generating a proprietary token or unique identifier, your service virtualization solution should have the ability to use scripts but not be limited to a single language. Different testers and developers use different scripting languages depending on the level of expertise or preference. At a minimum, your service virtualization solution should support Java, Jython/Python, JavaScript, and Groovy.

Since new message formats and protocols show up all the time, your service virtualization should also include a framework that allows you to extend the tool's capability. This will enable you to support any transport or protocol that your organization is using, whether it's an industry standard or custom implementation.

SUPPORT FOR AUTHENTICATION & SECURITY

Virtual services need to behave just like the real services, so that will include things like authentication and security. Your virtualization solution should give you the ability to validate incoming transport layer security such as SSL certificates, as well as interact with the live services during recording, which could be governed with authentication mechanisms such as oAuth, Basic Auth, Digest, Kerberos, NTLM, and so on.

Additionally, your service virtualization solution needs to be able to negotiate and emulate message layer security. Examples of this include payload encryption, SAML, signatures, and others. By being able to emulate the security mechanism, you can create the most realistic virtual services possible and ensure that any defects related to authentication or security are identified.

CONFIGURABLE PERFORMANCE ENVIRONMENTS

One of the most powerful applications of service virtualization is enabling performance testing. You can laser-focus on specific component SLAs by using virtual services to emulate out of scope dependencies. You can create performance environments by surrounding your application with virtual services. This allows you to do earlier stage performance testing and reduces your exclusive reliance on full performance environments.

In order to do this, your service virtualization solution must be able to adjust performance delays. You should be able to configure baseline performance delay times based on static values as well as be able to adjust performance delays dynamically as the virtual service is used (more hits = increasingly slower performance).

Additionally, it is valuable to be able to leverage your existing application monitoring profiles for your virtual services, so it's important for your virtualization solution to understand and import performance delays from solutions like App Dynamics or Dynatrace.

SUPPORT FOR CLUSTERING & SCALING

As your service virtualization solution becomes adopted by the performance team, or general usage increases, you will need your solution to enable scaling. Scaling can manifest in two forms: high throughput and fault tolerance.

To support scaling, you will want your service virtualization solution to be easily clustered behind a load balancer so that you can distribute the load given to your virtualization infrastructure appropriately, and your service virtualization solution should be able to maintain asset parity as you adjust different virtual services on individual nodes. This can be accomplished by coordinating your virtual services through a source control system or by having a centralized asset authority that deploys the appropriate virtual services in the right configuration to each node in the cluster.

OPTIMIZED WORKFLOWS

- RECORD & PLAYBACK
- AI-POWERED ASSET CREATION
- TEST DATA MANAGEMENT & GENERATION
- DATA REUSE
- SERVICE TEMPLATES
- MESSAGE ROUTING
- FAILOVER TO LIVE SYSTEM
- STATEFUL BEHAVIOR EMULATION

RECORD & PLAYBACK

Creating virtual services from actual transactions in your environment is the best way to build virtual services that emulate the real behavior of your live services. To do so, your service virtualization solution should be able to capture traffic from your environment, either natively or through integrations with solutions like Wireshark or Fiddler, enabling nontechnical users to capture their specific use cases into virtual services without intimate knowledge of the real back-end services.

Look for a service virtualization solution that gives you the ability to not only process the traffic, but capture that traffic into a service template, so that any decisions that you make during the traffic processing workflow can be captured into a reusable model.

AI-POWERED ASSET CREATION

If your service virtualization solution has intelligence built into the asset creation process, it will be able to make decisions about things you commonly do when creating your virtual services. An example of this is in automatically determining the appropriate transport, message, and data source correlation to apply when processing traffic.

Artificial intelligence allows solutions to analyze all of the available requests, looking for patterns and relationships, and then determining the best way to group that information while at the same time identifying and automatically generating a data source for the parameterizable values. With AI-powered asset creation, your solution will do the majority of the decision-making automatically, enabling nontechnical users to create virtual services quickly.

TEST DATA MANAGEMENT & GENERATION

Often the services that are identified as candidates for service virtualization are actually suffering from data challenges, so test data management and service virtualization go hand-in-hand.

In addition to creating virtual services, your service virtualization solution can generate the test data you need, tightly coupled with your test data management/generation solution so that you can capture data from your environments, mask that data for privacy reasons, abstract transactions into a data model, and then generate and subset data in those models.

Having a test data management solution that is deeply integrated with your service virtualization solution will allow you to tackle any virtualization opportunity with greater flexibility. For example, to create an early stage virtual service without the required data on hand, you can use your virtualization solution to create the virtual service from the service definition and then abstract that into a data model. You can then apply business rules into the data and generate the appropriate data to cover all of the various behaviors you are looking for.

Test data management can be a key enabler for your service virtualization solution, so having one that's easy to understand and adopt will ultimately help the entire virtualization initiative gain traction and provide ROI as soon as possible.

DATA REUSE

In the early stages of asset creation you may only be able to record a subset of the actual services behavior, so you should be able to rely on your service virtualization solution to incrementally add data to its data library. As you progress through the development cycles, you may want to add additional behavior and logic to your service without starting over from scratch every time. To facilitate this, you can choose a service virtualization solution that allows you to re-record new data and merge it into your existing data structures.

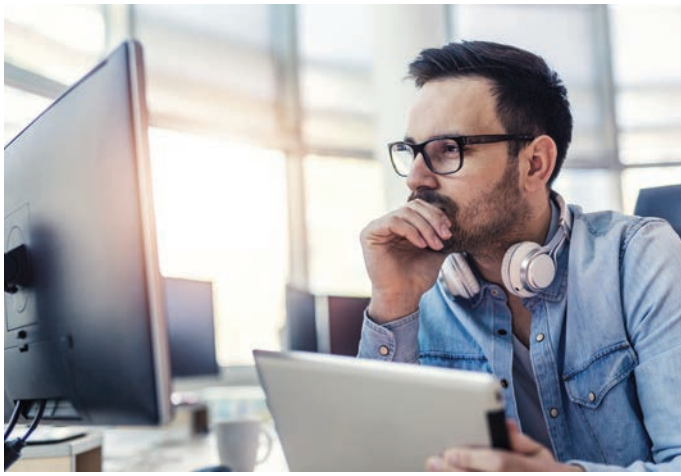
It is also worth investigating whether your service virtualization solution can incrementally update its data library via automation. Doing so will allow you to create a process where you capture additional data and merge it into your virtual services data library in an automated way.

SERVICE TEMPLATES

Service templates are critical to any service virtualization solution. Virtual services are like snowflakes — each one is different, and it can be really complicated for a centralized team to create virtual representations of an entire organizational service library. A key to a truly scalable governance process for service virtualization will be adopting workflows that allow you to incrementally maintain virtual services.

A service template can be thought of as an abstraction of a virtual service's creation logic, containing things like its deployment mechanism, data source connections, message correlation logic, data source correlation logic, any associated service definitions, data reuse strategy, target deployment server location, and so on.

By templatizing these configuration details into a reusable artifact, iteratively developing new behavior on top of existing virtual services becomes more manageable. Depending on the service virtualization solution you choose, service templates may manifest as different offerings, but ultimately you should look for a mechanism that allows you to abstract asset information into a reusable area that can be shared across the organization and updated.



MESSAGE ROUTING

Because virtual services are designed to process incoming messages and determine how to respond to them, that interrogation/determination process can be repurposed as a message router. There may be situations where you want to route certain messages to the live system, either because you want to capture some real behavior under certain conditions or you've decided not to virtualize pieces of your application.

You can also provide different virtual responses for different users by analyzing the incoming request and routing those messages to different virtual services. To enable this, your service virtualization solution should be able to repurpose any virtual service(s) into message routers, so you can appropriately funnel messages through your environment without having to reconnect applications all the time.

FAILOVER TO LIVE SYSTEM

Service proxies that you create for recording traffic can be repurposed as message routers as well. Your service virtualization solution can failover to the live system if your virtual service does not have the appropriate data. This will allow you to incrementally build your virtual service to cover common use cases at first, while still having all of the responses available even if you haven't captured them.

This technology can be used in the opposite direction as well, so you can create a "failover to virtual" scenario where the majority of your traffic goes to the live service unless that service goes down. At that point, the failover mechanism will allow you to funnel information to your virtual service so that when your environment is intermittently unstable, you're still covered by virtual services.

STATEFUL BEHAVIOR EMULATION

Stateful virtualization allows you to create virtual services that behave just like the real services and update themselves based on usage. Your solution can make state-based modeling easy to pick up by novice users, using simple interfaces to add the necessary logic to **C**reate, **R**ead, **U**pdate, and **D**elete data while using your virtual services. You'll be able to create more flexible virtual services and reduce the overhead of data management without all of that tedious mucking about in databases. This is extremely valuable when you want to build a process transaction flow like the ones often used in shopping cart applications and Open Banking API sandboxes.

SUPPORTED TECHNOLOGIES

- REST API VIRTUALIZATION
- SOAP API VIRTUALIZATION
- ASYNCHRONOUS API MESSAGING
- MQ/JMS VIRTUALIZATION
- IOT & MICROSERVICE VIRTUALIZATION
- DATABASE VIRTUALIZATION
- WEBPAGE VIRTUALIZATION
- FILE TRANSFER VIRTUALIZATION
- MAINFRAME & FIXED LENGTH
- EDI VIRTUALIZATION
- FIX, SWIFT, & OTHER PROTOCOLS

REST API VIRTUALIZATION

Your service virtualization solution should be able to emulate APIs using Representational State Transfer (REST). This includes support for service definition such as Open API (OAS 3), Swagger, or RAML. Your tool should be able to simulate URL, method, path, parameter, query, and JSON payload information as well as emulating headers, mime-types, attachments, and so on. Additionally, it needs to be able to consume and process restful requests, so the correlation can be applied on all of the message options listed above.

It is also important to be able to validate incoming RESTful requests, so you can accept or reject messages that do not conform to the appropriate schema. This is an example of using service virtualization as a powerful validation mechanism.

SOAP API VIRTUALIZATION

Simple Object Access Protocol (SOAP) is an interface that is still widely used in applications and is a good target for virtualization. Your service virtualization solution must be able to interface with SOAP APIs, including support for service definition such as WSDL and schema/.XSD. It must be able to respond to SOAP-compliant messages including SOAP action, attachments, WS policy, and relevant SOAP headers.

ASYNCHRONOUS API MESSAGING

Especially for reactive microservice environments, your service virtualization solution should be able to act asynchronously—send requests and responses without waiting for a corresponding reaction. An example of this would be creating a virtual service that upon invocation sends an asynchronous message somewhere else in your environment.

MQ/JMS VIRTUALIZATION

Middleware systems often cause a lot of trouble for our test environment and are therefore good candidates for service virtualization. To reap the benefits, your service virtualization solution must be able to simulate various queue/topic patterns, including point-to-point and publish/subscribe, to allow you to validate complete end-to-end scenarios and simulate systems that leverage these technologies.

IOT & MICROSERVICE VIRTUALIZATION

IoT and microservices are bringing a host of new testing challenges, including new complexities in our test environments, especially the ability to isolate individual microservices for testing. Service virtualization allows you to create the isolation you need for testing, so if you are using these technologies or may in the future, your service virtualization solution needs to be able to communicate over the interfaces specific to IoT and microservices (such as Websockets, MQTT, AMQP/Rabbit MQ, Kafka, and Protocol Buffers).

DATABASE VIRTUALIZATION

Database virtualization is arguably one of the quickest ways to get massive ROI from your service virtualization deployment. Databases are often bottlenecks in testing environments because multiple testers are interacting with

them and potentially polluting our data sources. Additionally, many databases may not contain the proper data or behavior that we are looking for given the different types of testing activities.

Simulating databases is a fast way to unblock an individual and give them total control of the test environment. To do so, your service virtualization solution needs to be able to intercept calls that are going to your databases, in order to record and create virtual services for them. Your service virtualization tool can also give you the ability to switch between live and virtual databases on demand. Coupling this with test data management is one of the most powerful ways to take control of your test environments and get maximum ROI from service virtualization.

WEBPAGE VIRTUALIZATION

An increasingly popular activity, webpage virtualization is a powerful way to provide training or demo environments to your organization and stakeholders. By backing a webpage with a virtual service, you can provide an experience through the web application that is customized to an individual user, specific demo flow, or to highlight new capability. Through virtualization you can simulate this without having to build all of the infrastructure in the backend. Your service virtualization solution can serve up HTML pages as well as be able to correlate RESTful information provided from interacting with the page, so that different pages can be served up through the users' journey.

FILE TRANSFER VIRTUALIZATION

Many legacy systems communicate by exchanging files, and they can become a bottleneck in your testing environments. As organizations move through the different maturity stages of service virtualization they often find that the legacy systems, which can be the most complicated, tend to provide the biggest ROI for service virtualization.

File transfers are an example of a communication mechanism that is difficult to emulate, so your service virtualization solution needs to be able to scan folders looking for files, pick up those files, process them, and then provide the appropriate response. This can either be a new file showing up in a different folder or potentially an asynchronous or JDBC call. This kind of process emulation is very valuable when you have a third-party system that you connect with, that communicates via file transfers.

MAINFRAME & FIXED LENGTH VIRTUALIZATION

For some industries in particular, mainframe virtualization can serve as a very powerful virtualization project. Mainframes are often out of the control of open systems developers, but hold a lot of the critical data needed to make these systems work. Communication to mainframes is often done through interfaces like REST, MQ, or TCP, and communication from mainframes to databases is often JDBC or DB2.

For mainframe virtualization, your service virtualization solution needs to be able to communicate over these mechanisms the COBOL copybook message format. This will allow you to approach mainframe virtualization initiatives and free up open systems development by decoupling mainframe dependencies.

EDI VIRTUALIZATION

EDI is a message format standard used to communicate business information between business entities. Businesses once used paper for these transactions (such as purchase orders, invoices, or in the healthcare industry, for instance, enrollment forms), which was extremely complicated and prone to error. To improve on the process, EDI was designed to standardize communications and make a "paperless exchange." Systems that communicate over EDI are great candidates for virtualization, so if you make use of EDI, your service virtualization solution needs to be able to provide a mechanism for sending messages in the correct dialect, version, and standard. Additionally, by combining EDI virtualization with file transfer virtualization, you'll be able to emulate legacy systems often found in insurance, finance, and medical industries.

FIX, SWIFT, & OTHER PROTOCOLS

There are hundreds of additional message formats and protocols to consider out there. This can be one of the hardest areas to clearly identify when choosing a service virtualization solution, but in order to fully cover all of the dependencies your organization may have in the future, it makes sense to take an inventory ahead of time so that you can understand the message formats and protocols that you will require.

To ensure you have the support that you need, make sure that your service virtualization solution has a custom extensibility framework, so that any unknown future or proprietary interfaces can be covered by creating a custom implementation.

AUTOMATION

- CI INTEGRATION
- BUILD SYSTEM PLUGINS
- COMMAND LINE EXECUTION
- OPEN APIS FOR DEVOPS INTEGRATION
- CLOUD SUPPORT (EC2, AZURE)

CI INTEGRATION

To dynamically deploy virtual services as a function of code check-in, your service virtualization solution should be able to integrate into your existing CI process. This will allow you to surround your application with virtual environments and execute your integrated test scenarios as early as possible, defining virtual service behavior such as specific data sources and performance profiles as a part of your CI configuration. This will allow you to deploy the right virtual services, the right way, automatically, and will greatly stabilize your CI pipeline.

BUILD SYSTEM PLUGINS

Many CI pipelines take advantage of build systems such as Jenkins, Microsoft's Azure DevOps, Atlassian's Bamboo, JetBrain's TeamCity, and many more. To optimize workflows, your service virtualization solution should have native plugins into these build systems so that you can accomplish your automation tasks that involve virtualization as a build step in your pipeline. This will not only make environment management a much easier task but will help build virtualization in as a part of your DevOps process.

COMMAND LINE EXECUTION

If your service virtualization solution can execute via command line invocation, you will be able to dynamically start and stop your virtual servers as needed when running your test cases. Your command line interface should be dynamic as well, so you can swap configuration details on the fly.

OPEN APIS FOR DEVOPS INTEGRATION

Open APIs that enable you to programmatically generate, configure, and deploy virtual services, will allow you to set up a client/server configuration for your DevOps pipeline and your service virtualization platform. A series of open APIs will provide you with the ability to set up a scalable infrastructure nd reduce overall licensing costs by programmatically making calls to the virtualization server from multiple areas of your organization as needed to configure the right virtual services on demand.

CLOUD SUPPORT (EC2, AZURE)

If your service virtualization solution can be deployed either on premise or into a cloud environment such as Amazon EC2 or Microsoft Azure, you will be able to swap out the underlying hardware with ease. Containerization is a big component of reference architectures as well, so look for service virtualization that gives you the ability to deploy the technology via Docker.

MANAGEMENT AND MAINTENANCE

- GOVERNANCE
- ENVIRONMENT MANAGEMENT
- MONITORING
- PROCESS FOR MANAGING CHANGE
- ON-PREMISES & BROWSER-BASED ACCESS

GOVERNANCE

As your service virtualization initiative scales, it will be important to set up a governance process around its usage, defining roles, responsibilities, access levels, policies, procedures, SLAs, naming standards, and more, so you can build a center of excellence that can handle a large volume of incoming virtualization requests without becoming a bottleneck. To enable this, your service virtualization solution should have mechanisms for defining best practices as well as utilization review, so you can identify who's using service virtualization and how, and audit the usage. A side effect of this is the ability to quantify your ROI through asset utilization, understanding the value that teams are getting from virtualization (and identifying the teams that have stopped using it).

ENVIRONMENT MANAGEMENT

Once you've built a large inventory of virtual services, your service virtualization solution should help you manage virtual test environments with a self service interface, so users can define what virtual services are connected to particular workflows, as well as what configuration is required to enable test automation of that flow. Instead of having to wait on a centralized team to deploy the proper virtual services, individual users will be able to select the appropriate virtual environments for their use cases, deploy them (on premises or into the cloud), and test away.

MONITORING

Successful service virtualization is based on users trusting that their virtual services are behaving the way that they expect, so monitoring is essential. DevOps engineers must trust that the proper virtual service has been deployed, and testers must trust that issues stemming from environments with virtual services are not causing false positives in their defect detection. Your service virtualization solution should enable you to monitor requests and responses that flow through your virtualization infrastructure, so you can identify errors out of the abundance of transactions that take place, and proactively identify and trace issues in your environments.

A PROCESS FOR MANAGING CHANGE

Many virtualization initiatives fall apart when it comes to API change. Users have spent a lot of time creating the necessary virtual services, and when the real services change things become out of sync. To solve this, your service virtualization solution can natively integrate with source control systems to enable maintenance of several versions of your virtual services for forward and backward compatibility.

Your service virtualization solution can also understand the API change by automatically mapping different versions of services to each other. Through this capability, users can create a change template that can be applied to impacted virtual services and automatically update them to the new version, maintaining data and logic in the service while still rapidly adjusting to the change. All in all, this will be one of the most critical capabilities you must have for true ownership of a long-term, successful service virtualization deployment.

ON-PREMISES & BROWSER-BASED ACCESS

Your ad hoc users will want browser-based access so they don't have to install anything, and your center of excellence will want powerful virtualization desktop clients that allow them to create the best virtual services possible. In order to facilitate this, you need to choose a service virtualization solution that allows you to do both, along with a centralized coordination platform that allows you to link these two types of users together into a collaborative architecture to prevent redundant asset creation.

SELECT THE TOOL THAT FITS YOUR TEAM'S NEEDS

Choosing the right service virtualization solution for your organization can be a huge undertaking. Moving away from industry buzz and focusing on critical features and capabilities that your organization will need is the best way to identify the right solution. To get a solution that has all these capabilities and more, discover Parasoft Virtualize . Learn more and get a free copy of the most powerful service virtualization solution on the market.

PARASOFT VIRTUALIZE

CREATE, DEPLOY, & MANAGE VIRTUAL TEST ENVIRONMENTS ANYTIME, ANYWHERE

When testing is at a standstill because systems are difficult to access, scale, or configure, you can rapidly create virtual test environments with Parasoft Virtualize. Use Virtualize to create, deploy, and manage simulated dev/test environments and minimize constraints that ordinarily arise from inadequate test data.

Unlike any other service virtualization solution, Virtualize can create realistic simulations by monitoring existing behavior, enabling users with limited expertise to quickly create reliable test environments.

TAKE THE NEXT STEP

Learn about the results Comcast achieved after successfully implementing service virtualization. Read the case study.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives — security, safety-critical, Agile, DevOps, and continuous testing.