With service virtualization, your DevOps teams can use virtual services instead of production services, enabling frequent and comprehensive testing even when key components are missing from your system architecture. By emulating the behavior of essential components that will be present in a final production environment, service virtualization allows complex applications to undergo integration testing much earlier in the development process, removing key bottlenecks that would otherwise delay production and time-to-market for an application under test (AUT). For the development of most enterprise applications, which rely on a mixed array of system components working together in harmony, service virtualization “fills in the gaps” of missing system components by simulating their responses to show how the various components interact. It is especially useful in the development of complex cloud-, API-, and SOA-based systems, as well as at any point in a production cycle where important hardware and software components aren’t readily available for testing purposes.

More and more companies are using service virtualization to improve productivity, reduce testing costs, and [deploy higher-quality software](http://smartbear.com/product/ready-api/servicev/features/share-virtual-services) in a shorter timeframe. In addition to emulating major software applications, third-party services, and even whole backend systems, the virtual assets can also be reliably shared and used by your entire production team, facilitating more efficient parallel development practices. By quickly and easily removing dependency constraints across your organization through virtualization, you can gain a competitive advantage over other companies still waiting in the linear-development limbo.

Why Use Service Virtualization?

Traditionally, testing teams have had to wait for nearly completed applications to be deployed before proper functional, integration, and performance testing could begin. Distinct project teams might produce different components of a system or application, one piece at a time, and then assemble them into a single working product before allowing the testers to have their way with it. It’s logical, it’s linear, and it’s slow.

Focusing on testing a nearly finished software application—with all of its components nicely integrated behind a functional user interface—will obviously remain an essential step in any development cycle, but in these days of rapid (and continuous) development cycles, it often isn’t practical to wait that long before testing to see how various software components communicate with each other. Testing needs to happen alongside development from the start, and this is especially true in the production of heterogeneous systems involving multiple layers of interdependent components, [third-party apps, and APIs](http://smartbear.com/all-resources/articles/what-is-api-virtualization/).

API layer, where major problems down the line are often first introduced into system interfaces, rather than having to wait to test a more complete production-ready app. Using service virtualization, developers can validate integrations earlier than would otherwise be possible. And considering that the user experience of an application is wholly a function of the sum of its parts, it makes sense to ensure that those parts are working well as they’re created, rather than waiting for a finished application. **Service virtualization can be useful at any point in the development of an application, from assisting with small manual unit tests to enabling automated performance tests of an integrated system**.

Unless your company has sufficient resources to provide your developers and testers with every actual component required by a production system, using virtual services throughout the development cycle can save a lot of time and money. Any software development team can benefit from service virtualization, especially when it isn’t practical to repeatedly test against dependent third-party components, such as Salesforce, Oracle, or PayPal. By virtualizing the behavior of a CRM, ERP, or payment gateway in your system architecture—with simulated data and software responses—your development efforts will be able to proceed freely and you’ll be able to conduct tests as often as desired, paving the way to smooth user acceptance tests down the line once the actual third-party components are deployed.

#### **Reduced cost**

Thanks to service virtualization testing, testing and QA costs have reduced by up to 67% in surveys and thus resulted in more allocations being made for development teams.

#### **Shorter time-to-market**

With improved testing capabilities, it takes less time for the product to hit the markets when compared to the previous scenario of having to wait for the QA teams to certify each and every component of the product to be market ready. With server virtualization, it becomes much easier to study the behavior of connected components in a demo environment than a live one.

#### **Improved Product Quality**

Service virtualization creates live replicas of actual product deployment scenarios and hence it is easy for QA teams to identify issues and failures that would have occurred when the product goes live for users. Thus the actual product that users get to use is one that is guaranteed to offer remarkable robustness and fault tolerance.

#### **Lesser constraints on QA teams**

QA teams do not have to wait for the product to be completely built by the development teams to facilitate integration testing. Virtualization opens up new avenues to test complex products in actual production environments.

When an organization is looking at “continuous everything,” an emerging best practice known as continuous testing is a critical component in the overall process. Another emerging best practice known as “service virtualization” enables continuous testing by providing anytime, anywhere access to a complete, simulated test environment.

Recommended capabilities for this goal include an intuitive interface for automating complex scenarios across the messaging layer, ESBs, databases, and mainframes, and touch on the following actions:

* Defining automated test scenarios across the broad range of protocols and message types used in APIs: REST, WADL, JSON, MQ, JMS, EDI, and fixed-length messages
* Automating rich multilayer validation across multiple endpoints involved in end-to-end test scenarios
* Parameterizing test messages, validations, and configurations from data sources, values extracted from test scenarios, or variables
* Defining sophisticated test flow logic without requiring scripting
* Visualizing how messages and events flow through distributed architectures as tests execute

With the trend of agile development and increasing system interdependency, it has become extremely difficult to access all of the dependent applications. Access to dependent systems and environments is required to execute the necessary type of complete end-to-end tests. By leveraging service virtualization to remove these constraints, an organization can gain full access to the test environment, thereby enabling continuous testing to occur as early and often as needed.

Service virtualization enables rapid iterative development by providing simulated test environments that can help scale continuous testing. The goal of service virtualization is to simulate interfaces and resources that may not always be available for testing due to cost or other constraints. This emerging industry best practice promises to provide a much more robust and comprehensive approach to ensuring that we can continuously deliver error-free cod

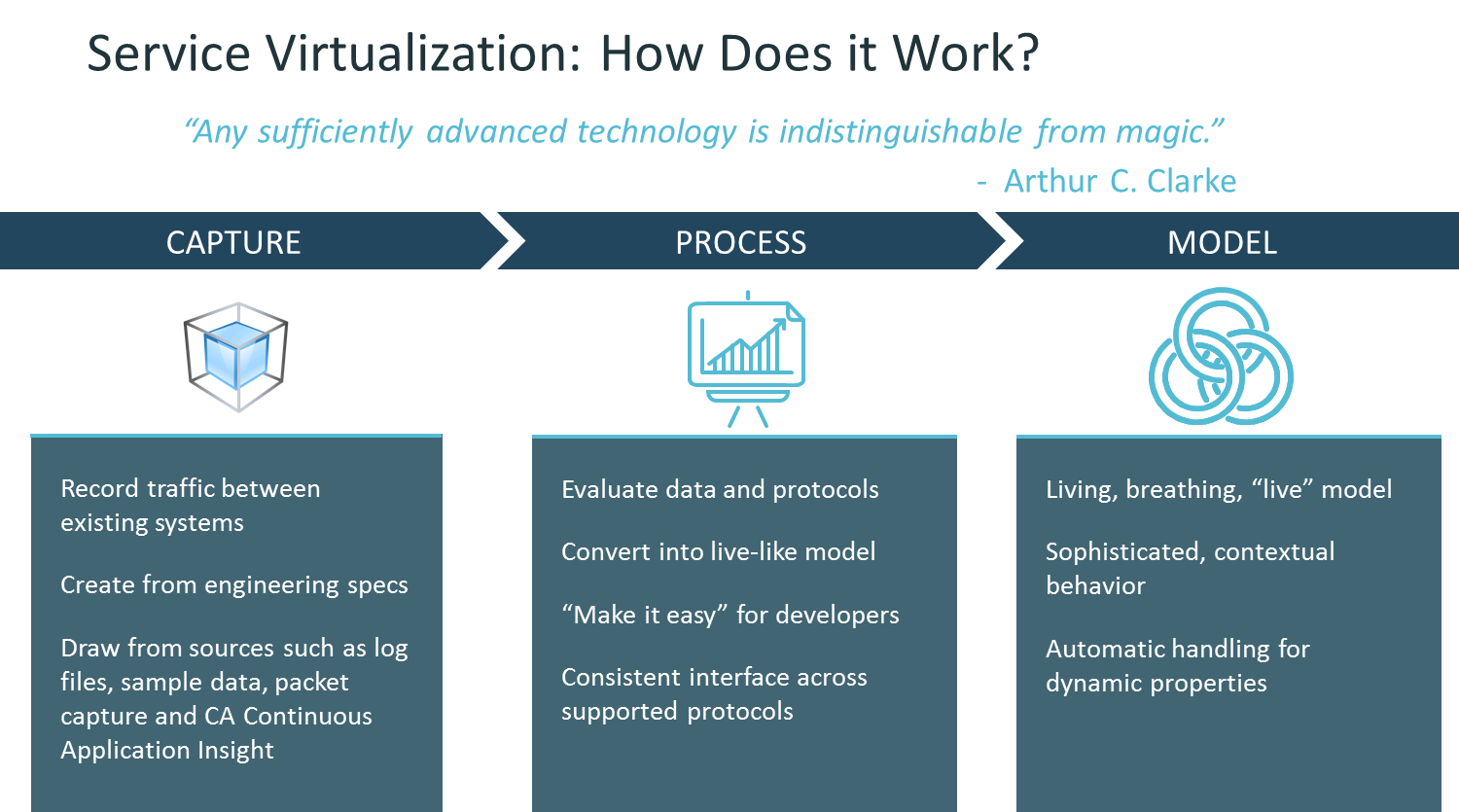
## ****Service Virtualization: What It Is and How It Works****

CA Service Virtualization solves access issues to third-party APIs and systems by capturing and modeling them as virtual services that act just like the real services. The virtualized versions provide alternatives that can be used for functional and performance testing.

When developers and testers use Service Virtualization, the services behave and perform similarly to the real thing without the underlying hardware and software complexity of a physical system. Development and testing continue just as they always have, but less constrained and without contention between teams for environments, labs, test data and so on.

Access to a virtualized API can actually be better than the real thing for the simple reason that you can test all kinds of scenarios — varying levels of functionality, performance and maintenance levels — with a virtualized service that you never could with an API. You can make the virtualized API behave any way you want. Best of all, multiple copies of the same API can be created simultaneously so that multiple people can use the API for testing at the same time.

Sounds like magic, right? Well, not really. This is how Service Virtualization works:

*[](http://servicevirtualization.com/wp-content/uploads/2017/01/CaptureProcessModel.png)*

### Step 1: Capture the Conversation

When software components communicate, they use a structured format and conversation, also known as a protocol. Within this structured conversation, observations can be made about static vs. dynamic elements, conversation content and data (the payload), and other aspects of the relationship between the components to create an understanding of the software interaction.

### **Step 2: Process the Captured Conversation**

Here, the Service Virtualization tool evaluates the requirements of the engineering specifications provided or analyzes the captured conversations between components. It is this processing and clever approach to the captured data that is what differentiates using a tool like CA Service Virtualization vs. stubs and mocks.

### **Step 3: Create a Usable Model**

CA Service Virtualization converts these captured conversations and processed protocol request/response pairs into a sophisticated, dynamic model that lives and breathes very similarly to the real thing and provides the scenario coverage and capabilities for software development and testing activities. The Service Virtualization tool is able to handle difficult tasks such as identifying dynamic data, stateful conversations requiring session IDs and other stateful techniques, and observe real-world variability and dynamic behavior of the conversations. After processing, CA Service Virtualization compiles a conversation into a stateful (or stateless, if desired) model. In this model, we now have the ability to handle challenges like state, automatic dynamic data processing, even populating responses with “fake” suitable test data, automatically solving a huge challenge common for customers around test data management. It is this compilation step that gives the model the rich, dynamic functionality required for realistic virtualization, and makes it not fragile and break-prone as developers use it in new cases and scenarios.

With Service Virtualization, developers have their own private environments for developing code. They don’t share environments and don’t need to wait for other third-party systems to be available, and they can test for performance without incurring higher costs from third-party partners.

With Service Virtualization, much of the testing at a component level can “shift left,” or be moved earlier in the SDLC. Because each component can be tested individually instead of waiting for complete assembly, unit and regression testing happens sooner, is more complete, and defects are identified long before integration or user acceptance testing. Finding bugs earlier means issues are fixed sooner instead of moving on to other projects before the defects are identified and costs to remediate are substantially higher.

Service virtualization emulates the behavior of software components to remove dependency constraints on development and testing teams. These constraints occur in complex, interdependent environments when a component connected to the application under test is:

* Not complete
* 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

SV can significantly speed up the development process in case required resources:

* are not available during (part of) the test cycle, thereby delaying tests or negatively influencing test coverage;
* are too expensive to keep alive (e.g., test environments need to be maintained or rented continuously even though access is required only a couple of times per year);
* cannot readily emulate the behaviour required for certain types of test cases;
* are shared throughout different development teams, negatively influencing resource availability.

CA Service Virtualization emulates (simulates) server side behavior so users can easily and quickly validate front-end functionality while CA Application Test is used to emulate/simulate client side behavior so users can easily and thoroughly validate back-end functionality.

TDM

Test data management is the creation of non-production data sets that reliably mimic an organization’s actual data so that systems and applications developers can perform rigorous and valid systems tests.

## What do I need to know about test data management?

The quality of test data matters. If applications are tested against generic data many problems can arise once the application is put into production. To avoid problems, applications must be tested rigorously against data that is as similar as possible to the actual data that will be used.

## Why not copy production data for tests?

Production data is often not practical for use in a test system due to security and regulatory concerns. Data that has personally identifiable information must be altered in order to protect people from having sensitive data exposed to the development and testing teams. Test data management uses data masking techniques to obfuscate personally identifiable information while still retaining the formatting and other data properties that are important for testing.

Test Data Manager allows you to do synthetic data generation. It gives you a high level of confidence in your data that you're creating. It also keeps you out of the SOX arena, because there's no production data within that environment. The more that you can put in controls and keep your data clean, the better off you are.

Our organization has a complex IT landscape. Each environment has different components, each holding a copy of accounts. With CA Test Data Manager, we are able to perform data obfuscation of personally identifiable information (PII) data, simplifying our data compliance related tasks.

CA Test Data Manager helps us reduce testing cost and speed up test cycles, enabling intelligent data masking, and providing reliable test data.

## Who uses test data management?

Test data management is used by organizations that do a lot of business critical processing of sensitive data. It is especially important in industries such as health care where a breach of sensitive customer data could be extremely damaging. However, most organizations have some data that is sensitive and needs to be masked for testing purposes.

## What are the benefits of test data management?

Test data management helps organizations create better quality software that will perform reliably on deployment. It prevents bug fixes and rollbacks and overall creates a more cost-efficient software deployment process. It also lowers the organization’s compliance and security risks.

Benefits of CA TDM tool (**Right Data, in the Right Place, at the Right Time)**

* + Reduces time-to-market with a complete end-to-end test data management solution.
  + CA TDM uniquely combines elements of data sub setting, masking and synthetic, on-demand data generation to enable testing teams to meet the agile needs of the organization.
  + TDM is is used to Build test data warehouse from production environment.
  + TDM takes data from database, creates repository, masks, profiles, provide data from ‘Data management’ data base

Improve quality. Providing the right data for testing can reduce defect creation by up to 95 percent.

Improve testing efficiency. Eliminate data constraints and reduce the time and resources required to provision data by 50 percent.

Reduce costs. Creating accurate subsets of data can reduce infrastructure cost by up to $50,000/database.

Key Features

Synthetic data generation. Create rich synthetic data which covers 100 percent of possible tests.

Data masking. Secure millions of rows of data in minutes using automated data profiling and high performance masking engines.

Data subsetting. Clone specific subsets of data into target environments.

Coverage analysis. Measure exactly how much of a system data can test.

Test data allocation. Reduce the time and resources to provision data by 50 percent using automated data discovery to request and receive exact data sets, linked to test cases.

Test data warehouse. Store data pools as reusable assets in a central repository and test multiple versions and releases in parallel.

Virtualization. Create realistic virtual data for service virtualization.

the quality of an application depends on the quality of the data for testing, and creating the right test data, to the right place, at the right time, remains a significant challenge. CA Test Data Manager can help organizations find, create and provision the data needed for testing—automatically and fast. Test data constraints, which routinely compromise application quality, can be eliminated, accelerating application delivery to market, on time and within budget.