ENTERPRISE VALUE INTEGRATION

EVI QUERY ENGINE

WHITE PAPER

ACCELERATING REVENUE RECOGNITION WITH ENTERPRISE VALUE INTEGRATION

ABSTRACT

Financial Accounting Standards Board (FASB and IASB) established ASC 606 and all companies will need to comply by 2019. This rule provides principles and mandates for organizations to recognize revenue from customers when a performance obligation stage is complete. In other words, which product or service is delivered in more than one step, revenue must be booked. This is causing panic among many organizations with complicated systems and contracts that were not written to reveal the value of each deliverable.

There are sixteen other vendors that have either produced a new module or have added new compliance features in their enterprise software to solve the problem. More often than not, they add complexity to organizations already dealing with multiple sub-systems. Enterprise Value Integration (EVI) is the only approach that offers a search engine to help people find information in the systems they already have.

This allows financial professionals to search for information contained in many different systems within a corporate network. They are able to see information that might be relevant, presented and ranked by a Recommendation Engine (RE). If there are multiple options, an accountant can find search the source record to determine which result is most relevant, then rank it "likely" or "possible." The user can find the contract, contractual obligations, transactions and initiate reconciliation from one search engine, then save the results to a profile of each customer. This integrates back to the sub-system in which each piece of datum is found. EVI Query Engine reduces days of searching and reconciliation to less than an hour per contractual obligation.

The more users work with EVI Query Engine, the smarter it becomes. It uses machine learning (ML) to improve results. ML and RE are both forms of artificial intelligence (AI). In this case, they are not used to replace any members of the accounting team, but to accelerate productivity.

It also lays the groundwork for running financial simulations to identify opportunities for organizational improvements with EVI methods. The simulations take information about the output of business processes that were used to satisfy each contractual obligation. We tap into either real cost accounting data from each expense or from the consensus of analysts around each business process. The simulations consider two variables: Shareholder Value and Customer Value. The output can have a billion or more outcomes, and the only question is whether it is worth additional investment to implement EVI in full or make precise surgical changes with the greatest impact.

PROBLEMS CREATED WITH REVENUE RECOGNITION REQUIREMENTS

Financial Accounting Standards Board (FASB and IASB) established ASC 606. This new rule mandates that companies report revenue from customers when a product or service is delivered in more than one step. For example, a computer may be ordered online, and the mouse and monitor are then delivered separately from the central processing unit (CPU). Another example would be a building. The first deliverable would be conceptual drawings from the architect; the second would be a finished design with three dimensional models to be presented to local planning and zoning officials; the third would be engineered working drawings for the general contractor; and the fourth would be the actual building, which required construction management. Currently, these may be billed and reported as the buyer and seller see fit. That changes with ASC 606 and must be implemented by all companies by 2019. Publicly-traded companies must comply by 2018.

ASC 606 has the potential to be highly disruptive for organizations on multiple levels: Contracts, philosophical differences, business processes, and disparate information systems

Contracts

Many contracts with customers are produced outside of the CFO's sphere of influence. They are often not written with a thought about revenue recognition. This will need to change in any organizations with contractual obligations met at different times.

Philosophical Differences

For more than a generation, finance and MBA students have learned about the theory of shareholder value. In the real world, this has devolved into a fixation on short-term share price. As a result, a CFO's team may use share price and other metrics which favor shareholders over customers. In investment decisions, these include internal rate of return (IRR) or net present value (NPR). This is often at the expense of maximizing long-term cash flow. And, those revenues comes from customers. This is a philosophical different than could be resolved during the course of the transition to complying with ASC 606.

Combined with EVI methodologies and algorithms, ASC 606 has the potential to resolve conflicts in philosophy, but it requires a more quantitative view of customer value, suitable for use by CPAs and other finance professionals. In Harvard Business Review, Anderson and Narus define customer value as, "what a customer gets in exchange for the price it pays." EVI defines it as what customers are willing to pay in exchange for something they value. Customers and consumers are similar. A consumer is presented with a product with a fixed price, and that price can be discounted, but is standard for all other consumers at that moment. Customers often feel free to negotiate a custom price based on their particular circumstances, as well as custom deliverables to meet their needs. For complex transactions, the seller or producer must now track when performance obligations are met at each step.

Business Processes

Those who work with business process management (BPM) are often associated with quality control and improvements, but don't have much or anything to do with contracts with customers. Maintaining a current inventory of business processes will enable an organization to track when it meets contractual obligations through the output of business processes.

There are many reasons to evolve contracts to specify performance obligations. EVI advocates for those obligations being defined not only as delivered goods or services, but the business processes used to produce the outputs used by the customer. The consideration for meeting each performance obligation is most often measured as a monetary transaction. If a customer is willing to specify the monetary value of each exchange in the contract, the company can satisfy reporting obligations of ASC 606 by tracking and reporting upon each set of business processes that enable the business entity to satisfy obligations. In doing so, an organization can deploy a number of advanced technologies to reduce the amount of manual time required to reconcile data from divergent financial subsystems. These advanced technologies include data mining, augmented intelligence (AI), machine learning, and service discovery.

Disparate Information Systems

If a company has multiple locations, it is likely to have a system for each country. Each is designed to address different circumstances, based on local currency, local regulations, and local practices. Some companies have separate data supply chain to inform risk management. Others have a system for procurement, another for accounts receivable (A/R) and yet another for accounts payable (A/P). It is also not unusual to also have a system for payroll, one for budgeting and another to manage assets. This creates a nightmare when it comes time to reconcile each transaction to comply with ASC 606. Presently, financial documentation is abstracted from the source data, and not designed to account for deliverables produced in stages. The immediate need is to find data in different systems that relate to each transaction in a semi-automated way, and present the findings to a member of the CFO's team.

RECOGNITION + RECONCILIATION

If a company has seven sub-systems, it can be very time-consuming to search for a single transaction and find related or supporting records in more than one of the subsystems. Many of these systems keep fragments of data, not in a form that is idea for manual reconciliation. They may also be entered into each system with different naming conventions. There is a need to introduce technologies that will automate the process. EVI adds-value by using this product as part of a migration path to optimize both customer value and shareholder value.

Step One: Mapping

The first step is for a company to work with the EVI team to identify each of the subsystems and provide access to each. The list of systems will appear in a user interface (UI). By default, each will be selected for a query. The user may unclick those systems not relevant to the query. For example, a customer transaction is not at all likely to be found the payroll subsystem.

Step Two: Integration

Once the systems have been identified, the data needs to be accessible. EVI's solution is to create an information portal. Other options for integration and access to data include data replication, shared business functions, distributed business processes and business-to-business integration. An information portal is most appropriate because it aggregates information from multiple sources into a single UI to avoid having the user access multiple systems for information – until they know where to drill down more deeply. Simple portals divide the UI into multiple zones, each of which displays information from a different system. EVI will present query results in the form of a spreadsheet, with rows and columns.

More sophisticated portals enabled limited interaction between zones. Other portals provide even more sophisticated user interaction and blur the line between a portal and an integrated application. EVI's solution is not to create an integrated application, but simply a way to show financial information most likely to meet the search criteria.

Given that much of the data is not easily found with existing systems, EVI is incorporating both data mining and a recommendation engine. This can show unexpected results, but also connect the dots between transactions that have been dis-integrated.

Step Three: Data Mining

There are two categories of data mining techniques: Classical and Next Generation. Classical includes Clustering, Statistics, and Neighborhoods. Rules, Networks, and Trees represent the Next Generation. EVI may incorporate an ensemble approach with more than one type of algorithm.

Step Four: Augmented Intelligence and Recommendation Engine

Raw query results may be "read-only," in that the purpose is to find relevant data, not to make modifications. The order of the query results will be prioritized based on Recommendation Engine technologies, also known as Recommender Systems, of which Netflix, Facebook, and Amazon are most famous. "Customers Who Bought This Item Also Bought" is a powerful way to segment buyers based on product selection, and also add incremental revenue during a live transaction. Steve Jobs is quoted to have said, "a lot of times, people don't know what they want until you show it to them." He was talking about consumers, but is it perhaps more true of a bookkeeper or accountant who has no idea how someone in another department may named or classified something.

Step Five: Confirmation or Correction

Once presented with ranking results, the user can then confirm whether certain results are relevant and to then take action based on that data. In other words, much like a old card catalogue at a library, an account will know where to look and can accelerate the process of getting to relevant data. If the user confirms, they will be given a list of Web Services that are most likely to match the transaction. In legacy monolithic applications, this will be convoluted or impossible, because tightly coupled architectures in these old systems often resemble a "black box." However, newer loosely-coupled systems are based on a multitude of components than can be identified. It is essential that the user confirm the relevance of specific services or microservices to enable two simulations and the second is a migration path. Simulations can produce millions or billions of possible scenarios, but need key data points. It needs a list of processes or services as well as transactions related to each. By pulling data from that AI system that has been used over a period of time, EVI Simulations will be based on hard data.

If results are less than optimal, the user can train the system to find more relevant results with a machine learning capability.

Step Six: Machine Learning

The system will improve query results based on machine learning (ML) techniques. These include Naive Bayes, Linear Regression, Logistic Regression, Decision Tree, K-Means, Random Forest, Support Vector Machine, and K-Nearest Neighbor. It is critical that learning from one person's query and correction be incorporated into the system to improve results of other users.

Once a member of the CFO's team is able to find relevant financial data, they have the ability to pinpoint where they need to go to reconcile transactions that comply with ASC 606. They also can participate in a concerted effort to relate financial transactions to the operational systems, which include legacy monolithic applications all the way to modern microservice architecture (MSA).

MIGRATION PATH

To migrate from the current state, enterprises will need to identify as many relevant business processes as possible and catalogue them. The EVI team has already worked with the National Retail Federation (NRF) to build a library of standards for retailers. The team will continue to work on building or adding-value to existing standards in other industries.

Index of Processes and Services

As a first step, it necessary to correlate each business process to each related web service or set of services. An index of services requires registration and discovery, which are already being done on cloud-based systems. Diversity of systems present seemingly insurmountable challenges for companies that wish to manually catalogue or index their processes and services, if they have not already done so. EVI strives to automate the process wherever possible.

Many companies have migrated partially to a service-based systems architecture, but still have legacy or monolithic applications. Monolithic applications are very different that contemporary architectures. Service Oriented Architecture (SOA) or Microservice Architecture (MSA) are defined by business processes, and execute them. Services may be hosted on premise, within a private cloud, or public cloud. Hybrid systems are common and also present additional challenges in creating an index.

History: Then and Now

A transition away from mainframe computing led to the use of Web Services, which connects machines together based on a specific function. Gartner played an important role in creating awareness and demand for Web Services in the late 1990s. Both public and private registries and exchanges emerged. Some of the exchanges were free and other entailed costs to access Web Services. Usage increased and Gartner forecast that four in ten financial services transactions would use Web Services models by 2004. At the time, IBM, Software AG, Sun Microsystems, Oracle, and Microsoft produced development tools to serve this demand. In particular, Microsoft introduced My Services for consumers to access Web Services in 2003, following the protocol known as Universal Description, Discovery and Integration (UDDI).

UDDI was one of many methods that conformed to a protocol known as remote procedure calls (RPC). Others include Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (Java RMI), and Distributed Component Object Model (DCOM). Sometimes known as a Subroutine Call or Function Call, RPC enables an application to request a Web Service without having to understand the network's details. It makes it easier to access a function or subroutine on a network on another computer.

Then: Client-Server Model

RPC follows a "client-server" model. The application is a client, which sends a request to another application on the server. It sends a response back to the client in short order. Tech Machina observes, "RPC-style Web Services are often implemented by simply generating services interfaces that map directly to existing language-specific function or method calls." The blog points out that the brittle and complex interfaces with need to be continually regenerated make the object model inherently unstable.

Now: Document-style

A more stable approach to Web Services has emerged in the form of what is known alternatively as Document-style, Service-Oriented Architecture (SOA) style or Messaging Model. This is seen as some as a radical approach, because it emphasizes exchanging messages as well as the schema. Web Services Descriptive Language (WSDL) provides a contract for each document, and results in fewer dependencies. This is departure from a rigid set of operations, as represented by RPC. It is rigid because it is not consistently interoperaable with other platforms. RPC is also is less secure as data is passed through Internet firewalls.

Microservice Architecture (MSA) is the embodiment of Document-style. Martin Fowler is one of the leading experts on MSA. He asserts that "automated deployment, intelligence in the endpoints, and decentralized control of languages and data" are the common characteristics of a microservice. These breakthroughs fall into the category of loosely-coupled architecture that minimize failures as a result of dependencies. If you think of these MSA components as independent bricks, the mortar consists of either networking or APIs.

The advent of MSA has created significant overhead in a much more dynamic environment. Resources are required to perform tasks, including memory, bandwidth, and computation time. Overhead is any combination of excess or indirect resources. Benjamin Wootton explains, "all of these services potentially need clustering for failover and resilience, turning your single monolithic system into, say, 20 services consisting of 40-60 processes after we've added resilience. Throw in load balancers and messaging layers for plumbing between the services and the estate starts to become pretty large when compared to that single monolithic application that delivered the equivalent business functionality!" Wootton is CTO of Contino.

Then: Fewer Endpoints with Web Services
Before MSA, systems typically had dozens of Web Services with well-defined endpoints. An endpoint can be a directory, folder, a path, or URL and can be inbound, outbound, or both.

Now: Many Endpoints with MSA
With MSA, there can be thousands of service endpoints, which require the use of virtual image processing (VIP) libraries – designed to process large images. "Libvips" are considered to be mature and well-documented, but is not well-known outside of a small circle. A VIP is a threaded library with no image size limits and fully demand-driven.

Then: Hundreds of Backends
Before MSA, there were long duration backends with hundreds of implementations. A backend has a specific role or responsibility in a system and maintains business rules or data relevant to a certain domain.

Now: Thousands of Backends
MSA can have more than a hundred thousand backends spun-up in miliseconds, and destroyed just as quickly as demand fades.

Then: DMZs

Before MSA, there were DMZs and firewalling to create a secure infrastructure. A firewall is a first line of defense architected to protect connections and network traffic. It is often built with both hardware and software to examine each message packet and filter out those that do not meet the specified security criteria. It does not authenticate individual users. DMZs are based on the geopolitical term “de-militarized zone,” such as the land between North Korea and South Korea. An administrator might put public-facing servers in the DMZ network, which can be remotely rooted. This separates what the public can access, from the authenticated users. Having public and private environments often requires two firewalls, but doesn’t necessarily address the issue of authentication.

Now: Handshakes

Since MSA, there can be thousands of services requiring authentication (Auth) handshakes. There are many Auth handshake techniques to authenticate the identity of those making queries.

Prabath Siriwardena feels that an abstract class known as JSON Web Token (JWT) is the best practice because it defines a container to transport data between interested parties and uses a JSON Web Signature (JWS) and an encrypted JWT, also known as JSON Web Encryption (JWE). As noted JWT is an abstract class and JWS or JWE are concrete implementations. Siriwardena notes, “the user context from one microservice to another can be passed along with a JWS.” Trust between microservices can be established with trusted certificates to each microservice with root certificate authority or an intermediary. This reduces overhead, says Siriwardena, who is senior director of security architecture at WSO2.

Then: Enterprise Service Bus

Historically, the software vendor and integrator MuleSoft has specialized in Enterprise Service Bus (ESB) but it has evolved and expanded its offering to enable connectivity with this layered approach, in which end-user experiences are provided by Experience APIs, Process APIs orchestrate systems, which are exposed through Systems APIs. Layered APIs shares things in common with SOA, such as being fine-grained. One big difference is that multiple network calls can be made through APIs layers, which is not true of SOA.

Now: MSA Patterns

Barbara Liskov describes a pattern as “a standard solution to a common programming problem.” By looking at the problem first, the appropriate implementation structure can be applied. It can include connections among program components or the shape of an object diagram or object model. Liskov is author of Program Development in Java.

Six of the most common microservice patterns are Message-Oriented, Event-Driven, Isolated State, Replicating State, Fine-Grained (SOA), and Layered Application Programming Interfaces (APIs).

Web Services and Business Processes were once complicated by the issue of State. By their nature, Business Processes are Stateful, in that changes occur after each step is performed. The moment of each event is measured as the clock ticks. In the early days, Web Services were always Stateless. This was slowly resolved with non-proprietary solutions such as business process Management Notation (BPMN) and Business Process Execution Language (PBEL). Yet, some Web Services execute or expose computing functions and others are executing business processes. In some cases, a clock matters and other cases, it does not.

Kristopher Sandoval sees a reason for people to be confused, noting "stateless services have managed to mirror a lot of the behavior of stateful services without technically crossing the line." His explains, "When the state is stored by the server, it generates a session. This is stateful computing. When the state is stored by the client, it generates some kind of data that is to be used for various systems – while technically 'stateful' in that it references a state, the state is stored by the client so we refer to it as stateless. Sandoval writes for Nordic APIs.

Stateful Patterns and EVI

Traditional system design favors consistent data queries and mutating state, which is not how distributed architectures are designed. To avoid unexpected results or data corruption, a state needs to be explicitly declared or each component needs to be autonomous. Event-driven patterns provide standards to avoid side-effects of explicitly declaring a state. Message-oriented systems use a queue, while event-based also sets and enforces standards to assure that the design and behavior of messages over the queue have a timestamp. A materialized view of the state can be reconstructed by the service receiving it. It can then replay the events in order. This makes the event-based pattern ideal for EVI. However, any pattern that records time stamps are suitable. Therefore, an index of microservices should also attempt to classify whether a Service has a time stamp, using the input of whether it is stateful as a key predictor.

To start, "service discovery" is required. The resulting index will then be the foundation for assisting the CFO's team in complying with revenue recognition mandates. It will also allow companies to start writing contracts in new ways, based on an inventory or Business Processes and Microservices. The eventual goal if for event-based accounting systems to correlate each customer transactions with each business process.

Service Registration

Registries of service instances already exist. However, it they typically reflect the current state of active instances. The network location is registered at the point a service is spun-up. A service is often described as ephemeral, because it is deleted from the service registry as soon as the instance is terminated.

Netflix Eureka is an example of a registry using a Representational State Transfer (REST) API. A service instance can be registered with its network location using a POST request. A PUT request refreshes its registration every 30 seconds. A client can querying service instances using HTTP GET request to access registered service instances. An instance registration may time out or an HTTP DELETE request can delete it. This illustrates the fleeting nature of services, and the importance of being able to create an index of inactive services over time.

There are two main patterns for registration: self-registration and third-party registration. Eureka is an example of a service instance registering and deregistering itself with a registry. An example of a third-party service registry is the Registrator project, which is open source. Service instances deployed as containers are automatically registered and deregistered. It is designed specifically for Docker containers and supports ETCD and Consul.

With the third-party registration pattern, the service registrar monitors and records changes to the set of running instances. It does this in one of two ways. The registry can subscribe to events or poll the deployment environment.

Therefore, a master index of names of all services is possible, but requires continual monitoring to index or catalogue the names of each and begin to associate each with its state and see a pattern or network locations used over time.

Service Discovery

The common use case of service registration revolves around real-time discovery of locations of services on the network. It is ephemeral. Some service instances are relatively static and others are dynamic. Network locations of instances are more static in an application running on physical hardware. A system can use configuration file to identify the network locations in static environments. These files are updated periodically, and can be easily incorporated into the index of services. Nginx points out that Service instances have “dynamically assigned network locations. Moreover, the set of service instances changes dynamically because of autoscaling, failures, and upgrades. Consequently, your client code needs to use a more elaborate service discovery mechanism.”

Data can be transmitted in a service-based environment in the form of a datagram or packet. In the most common usage, service discovery enables awareness of instances of each process in the cluster as it listens via a specific User Datagram Protocol (UDP) port as well as one for Transmission Control Protocol (TCP). UDP is commonly used by applications with real-time requirements. UDP can prioritizing on-time delivery of most packets, while dropping others along the way.

Jeff Lindsay is a programmer and blogger. He notes that service discovery, "is increasingly gaining mindshare in mainstream system architecture. Traditionally associated with zero-configuration networking, its more modern use can be summarized as facilitating connections to dynamic, sometimes ephemeral services." Lindsay explained that increasingly dynamic compute environments are becoming more common. Microservices and containers are important advances that also complicate service discovery. New solutions are being developed to solve new problems.

Log Files

EVI takes discovery in a different direction by using past records to maintain a running inventory. Log files contain time stamp and data about internet protocol (IP) address, URL that referred user agent, access request, username, quantifies number of bytes transferred, and lists the result status. EVI aggregates each log file, which retained by the servers.

TOOLS AND TECHNOLOGY IN SERVICE DISCOVERY

On-premise legacy systems cannot easily scale and do not have the flexibility to enable rapid business model innovation. Virtual machines (VMs) opened the door for greater flexibility, not confining a company to a finite set of servers. Containers take that to an entirely new level because they can enable your organization to pack a lot more applications into a single physical server than a VM because they consume less system resources. Containers are also faster because they don't contain a full copy of the operating system (OS) when they spin-up. VMs also use more random access memory (RAM) and central processing unit (CPU) cycles. As administrators have realized containers enable their teams to create a portable, consistent operating environment for development, testing, and deployment, the Docker and Kubernetes have become increasingly popular.

Docker & Kubernetes

Docker has a free open source and paid enterprise versions and is considered a simpler and more flexible container storage platform than Kubernetes. Although it is also open source, it comes from Google and is considered more intricate. Some say overly-complicated.

The simplicity of Docker is illustrated through service discovery. As long as the container name is used as hostname, a container can always discover other containers on the same stack. Docker Cloud uses directional links recorded in environment variables to provides a basic service discovery functionality.

The complexity of Kubernetes is illustrated through service registration and discovery. Kubernetes is predicated in the idea that a Service is a REST object. This means that a Service definition can be POSTed to the apiserver to create a new instance. Kubernetes offers Endpoints API, which is updated when a Service changes. Google explains, "for non-native applications, Kubernetes offers a virtual-IP-based bridge to Services which redirects to the backend Pods." A group of containers that deployed together on the same host is a Kubernetes pod. Kubernetes supports domain name servers (DNS) discovery as well as environmental variables. Google strongly recommends the DNS approach.

Smart Stack

Airbnb needed a solution that addressed its service discovery needs. The hospitality broker between hosts and guests did not think that discovering DNS were sufficient for its needs. DNS suffers from propagation delays. Exact propagation delays are typically non-deterministic because various layers of caching in the DNS infrastructure. DNS cannot be used to push state, so the consumer have to poll for all changes.

So, Airbnb built is own framework. It then posted it on Github in the public domain. SmartStack automates service discovery and registration. It handles creation, deletion, failure, and maintenance work of the machines running code. It has two primary components called Nerve and Synapse. Nerve tracks services in a centralized automation center and uses Zookeeper as its key-value store. Synapse discovers remote services.

Mesos

Mesos is part of the Apache project. It provides scheduling and resource management in cloud environments and data centers. The cloud promises scalability and Mesos was designed to scale to tens of thousands of nodes. One drawback of Mesos-DNS is that it is stateless, which interferes with integrations is business processes. Mesos was built at a different level of abstraction than the Linux kernel, but uses the same principles. Mesos is well-suited to provide service discovery functionality when used with Zookeeper, which is also part of Apache project.

Zookeeper

When used under the proper conditions, ZooKeeper maintains configuration information, discovers services and presents them in a simple interface to a centralized coordination service.

For the purpose of EVI's Service Discovery, we do not need to deploy most of what Zookeeper does in the deployment phase. The Zookeeper blog states, "each time they [applications] are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed." Zookeeper includes key/value stores and libraries.

ETCD, Consul and More

Other service discovery tools such as ETCD, Consul, Spartan, Marathon-lb, and Minuteman include additional functionality. ETCD also has a representational state transfer (REST) application programming interface (API) and builds consensus using RAFT. This is used as part of cluster of nodes that maintain a replicated state machine, while sending and receiving messages in the Protocol Buffer format. RAFT is actually a consensus algorithm for managing a replicated log and represents and alternative to Paxos. Ongaro and Ousterhout not that RAFT "separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered." Diego Ongaro is lead software engineer, compute infrastructure at Salesforce and John Ousterhout is professor of computer science at Stanford University.

Consul has a REST API as well as domain name servers (DNS). Alone, DNS service discovery is basic, and maintains A records. Address or "A" records are one of the primary records used in DNS servers, but it maps a domain name to the internet protocol (IP) address (IPv4) of the computer hosting the domain. For example, if you enter domain.com, the A record might look like 202.34.77.253. Adoption of IP version six (IPv6) is increasing because it has protocol enhancements for security and has a much larger address space. IPv4 has four sets of numbers, while IPv6 has eight. A "quad A" is used as an alternative to the A record with IPv6 addresses. Quad A stands for AAAA record.

In a dynamic, ever changing environment, Zookeeper propogates state updates with ZAB. It is a broadcast protocol. SkyDNS is similar, in that it is a distributed service to announce services, and is built on ETCD. It translates ETCD keys and values to the DNS.

Discovery Patterns

One problem is that many organizations have systems that also include legacy applications, but there are patterns that can be used to ameliorate this challenge. The administrator needs to determine which pattern fits their need based on their circumstances. There are four common discovery patterns: Self-registration, third-party, client-side, and server-side patterns. For microservices, there is also an anti-pattern with server-side discovery.

Zookeeper also supports the client-side discovery pattern with legacy applications. Network locations discovered by the client. The client launches or triggers the connection in TCP, and the server listens and accepts the connection. Commonly, servers are software program operating remotely, and can be accessed from a user's device, such as a mobile device or workstation. A client is a computer application that operates on a user's device and connects to a server. On the internet, a web server can be accessed from a web browser, which is a client. With Zookeeper's client-side discovery pattern, there is an integrated library, and the client performs load balancing. Zookeeper is written in Java and provides Java and C language bindings. In general, client-side discovery provides a service registry, which is independent of the API Gateway and the Microservice it supports or exposes.

EVI does not need to draw on the load balancing functionality. EVI must offer value by extending the library based on a client-side pattern, when there are monolithic legacy applications.

In contrast, the server-side discovery pattern runs a query to a service by way of the load balancer. The load balancer is aware of the service, not the client. The load balancer is dynamic and often uses DNS service registries in real-time. Server-side discovery can also substitute the load balancer with an API Gateway. This connects to the Service Registry as well as a Microservice. Both are independent of one another, but share access to the API Gateway.

Workday is one of the most innovative vendors of financial applications with a software-as-a-service (SaaS) business model. Workday is deploying software-defined compute, networking and storage, within an Amazon-like availability zones datacenter architecture. It uses Docker deploy and scale images on instances across distributed systems, but started with OpenStack to make the transition to MSA.

OpenStack

Endpoints, backends, secure infrastructure and finding ways to reduce overhead are critical in the migration to MSA. OpenStack is a cloud operating system (OS) that manages compute, storage, and networking resources throughout a datacenter. It provides administrators with control, yet also distributes responsibility to users to provision resources through a web-based dashboard. OpenStack is both an infrastructure and platform layer and utilizes Magnum and Murano. Magnum is an infrastructure management service and Murano is an application catalog. Yet, OpenStack does not have a fully formed service discovery tool, although components such as Keystone, Nova and Neutron all offer ways to manage and catalog components and services, especially with containers. OpenStack also permits Mesos, Kubernetes, and Docker to run on top.

There are third parties attempting to close the gap with OpenStack. NGINX Plus works with OpenStack Heat to offer a simpler DNS-based service discovery to dynamically discover and update the IP addresses of backend servers via the DNS protocol. OpenStack also permits Mesos, Kubernetes, and Docker to run on top.

TRANSITIONING TO CONTAINERS AND CONSOLIDATED FINANCIAL DATA SUPPLY CHAINS

Legacy financial systems reflect decisions of the past, which were often informed by limitations at the time. Multiple financial subsystems are one of the most daunting problems that will create havoc, as ASC 606 is formally mandated. In the meantime, cloud computing has moved forward rapidly and operational computing has become more removed from financial systems. EVI strives to close this gap, and use Workday as a starting point.

Workday's Approach

With a multi-tenant cloud delivery environment, finance users access and the latest version, with semi-annual updates that are automated. This is typical of software-as-a-service (SaaS), but a revolution for finance users. Existing configurations are preserved, while they face new regulations and rules. The system emphasizes modern business processes, and the ability to change them as conditions warrant. The Pleasanton, CA-based vendor provides a centrally managed business process framework. These can be applied to all activities in a business, but have been most heavily used by those in human resources, performance management, payroll, and finance in client companies. Workday customers include Bain Capital, Christiana Care Health System, Georgetown University, Trip Advisor, Best Western and Netflix.

Founders Dave Duffield and Aneel Bhusri felt that traditional relational client-server architectures were no longer addressing the needs of enterprise users. Duffield was the founder of PeopleSoft.

Workday decided to take a new approach at every level of its architecture, inspiring Workday made four key decisions at its inception. These visionary choices have allowed Workday developers to focus on business-logic development First, it moved away from a relational database design or applications to object-oriented. Second, it moved away from disk storage using SQL to store application data, to make the data available in-memory using random access memory (RAM). Third, it moved away from a user experience based on the limits of systems and user interfaces to an experience generated by users. Fourth, Workday it moved away from coding to definitional, metadata development. Metadata is the basis for Worktags.

"Worktags" enable financial users to identify vital data about business events: who, what, where, when, and why. This is a manual way to build consolidated event-based accounting system. William McCarthy first developed Resources-Events-Agents (REA) in 1982, and Workday and brought it into production with Worktags by capturing more detailed data from the start of a process. This not only enables more fine-grain insights to emerge, but also accelerates reporting at the end of each period. Workday's Mark Nittler explains, "a tag could also be a department, customer, product, supplier, or subsidiary, while other types of events include a generated invoice, a submitted expense, or a payment received. The ability to tag these businesses attributes to events—either at the time they occur or a later date if needed—is the building block in providing our Financial Management customers with a complete, multi-dimensional picture of their operations." Nittler is vice president of enterprise strategy at Workday and a certified public accountant (CPA).

REA is designed for modern computing, so it does not look like other financial systems. Oracle, which used Activity Based Costing (ABC) can have 2,161,603 columns in its database, which is very inefficient. Dennis Howlett sums it up, "Workday has three database tables but makes no use of relational database management systems (RDBMS) capabilities. Most everything else for both its human capital management (HCM) and now its financial applications are held in a combination of 423 classes and around 40,000 methods inside its Object Management System." Howlett is a contributor to ZD Net and formerly with CFO Magazine.

Workday's Nittler is a big proponent of REA and cloud-based architecture. "It turns out that the source of many of traditional software's challenges was found in the design at the very heart of those systems—the way transaction data was modeled. The origin of much of the pain is the code block, which is intended to provide a consistent way of identifying data needed to create journal entries from business events. Today, this code block is at the root of the most challenging problems of traditional financial systems. A limited number of fields are available, enough to create accounting but woefully inadequate to provide a multi-dimensional management view of a business. Once the code block is established it becomes like concrete, brittle and very difficult to change."

Built with resilient and reusable microservices, Workday leverages generic components. Each customer is a tenant in the distributed MSA architecture. Tenant's available distributed compute services are provisioned based Service Discovery. When there is a new request by a user, Workday interacts with this service and determine if it is possible to route it to an existing compute service or if it needs to dynamically launch a new one.

Workday exposes public web services APIs, as well as REST APIs.
Workday offers an open, standards-based API for programmatic access to business services with the public web services API. Users generate interactions using REST API. The self-service model quickly return a small set of frequently used data.

Workday and Docker
EVI will begin with a focus on Docker because Workday uses **Docker.** Service discovery and load balancing are two critical considerations for Workday and its approach to microservices. Workday and other developers utilizing MSA consider and design how each service is being discovered by other services within or from outside the cluster. Services should be equally utilized for maximum load distribution as they scale horizontally across the cluster. Scalability is the basis for Docker Universal Control Plane (UCP). It is available as part of Docker Datacenter, which also includes Trusted Registry and Commercially Supported Docker Engines. Docker Datacenter is an open source

GitHub project: https://github.com/drewkhoury/docker-datacenter

The **Registrator** project inspects Docker containers as they come online and automatically registers and deregisters services for any container. Developed by Glider Labs, it is built with pluggable service registries, which includes Consul, SkyDNS 2 and ETCD. Therefore it is interoperable. “Whether using containers or not, a service will always boil down to a long-running process, and a process may listen on several ports. This could imply multiple services,” explains Jeff Lindsay. “One could argue that if a process listens on multiple ports for the same functional service, it might be a good idea to collapse it into a single service. In whatever form it comes, a coprocess comes with two challenges: configuration and manageability.” Lindsay is the founder of Glider Labs. He points out distinctive properties of Registrator. It’s automatic, and doesn’t depend on the cooperation from within the Docker container because it uses container introspection for good defaults. Generic metadata is an environment variables used to define the services, which happens during container authorship, as well as at runtime.

Lindsay envisions the possibility of the metadata Registrator being used as a common interface for automatic service registration outside of just Docker. Registrator is an open source

GitHub project: https://github.com/gliderlabs/registrator/

An alternative is **ContainerPilot**, which is dependent on Consul. It registers a service with Consul on start and periodically sends time-to-live (TTL) health checks to Consul. ContainerPilot will no longer consider the application node active when the service fails to respond to the health check and once the TTL expires. It also monitors changes in dependent services when it polls Consul. It follows Autopilot patterns, which were developed by Joyent. Autopilot automates code for operational tasks of an application, such as startup, shutdown, scaling, and recovery. It is designed for microservices applications with any number of components, but can work in a single container. It is also an open source.

GitHub project: https://github.com/joyent/containerpilot

There are additional projects related to Docker and service discovery. These are a good place to start to understand the challenges and opportunities.

CONCLUSION

By optimizing to two variables, EVI is able to create much more value for customers as well as shareholders. In order to do this, what a company actually does needs to be considered in realistic terms. That is, less abstract and more concrete. The steps in each business process represent the most finite and discrete component in a whole system designed to generate value. By itemizing business processes, they can be analyzed, measured, redesigned and improved over time.

EVI is designed to create more value by aligning all the things a company does with creating value for customers. Contracts can spell-out which business processes are being used to deliver customer value, and quantify dollar amounts of each. Customer value is what customers are willing to pay for and enterprises have a responsibility to measure what they will pay, and make system work together is the simplest way to contain costs and confusion. Microservices and financial systems are at the nexus of this initiative. And the bridge between the two is service discovery. Knowing what each service does is the first step to measuring the costs associated with each and taking steps to make the organization more profitable over the long-term.

Contact: Stephen Turner
Practice Leader, Enterprise Value Integration
stephen@enterprisevalueintegration.com
415.726.8433