The imperative to become cloud native

A call to action for retail and consumer product companies

Introduction

Retail and consumer product companies are dealing with a complex set of challenges that point toward deeper adoption of the cloud. E-commerce and changing consumer demographics and preferences have led to store closures and erratic sales growth. At the same time, companies need to balance a higher focus on performance with ever-tighter needs for cost efficiency. That's why retail and consumer organizations should extend cloud adoption and become "cloud native" in their core systems and technologies.

Operating in the cloud is about more than moving from one platform to another. Imagine not having to sweat through scalability during the holiday rush—because the capacity you need is there on demand, all year round. Imagine being able to add impressive new omnichannel capabilities without busting the IT budget—and as a result, addressing the fact that 56 cents of every dollar spent in a store is influenced by a digital touchpoint. Or imagine slashing 90 percent of the time and effort it takes to analyze and predict customer behavior based on daily store activity.

Going cloud native is a key element in a retail and consumer product company's drive to achieve scale and growth. Most of these organizations are already using the cloud. This step is the logical continuation of that journey.

The cloud native approach relies on established cloud design patterns and uses prebuilt cloud provider capabilities, reference architectures, and standards to help build independently scalable, resilient, flexible, efficient, and distributed applications.

No matter what cloud commitments are already in place, this is the time to consider accelerating to a cloud native footing. The availability of competing cloud providers, open-source technologies, new cloud-based building blocks, and energized talent makes it possible to achieve significant agility, cost-competitiveness, and sustainable business advantage.

Background

Moving on-premise workloads to the cloud is becoming more prevalent as a way for consumer-facing companies to enhance efficiency. By one estimate, moving applications to the cloud can reduce operational costs by more than 25 percent. The attractive flexibility of the "pay-for-what-you-use" business model is driving them to explore the multiple hosting and financial consumption models of cloud hosting providers such as Amazon (AWS), Google (GCP), and Microsoft (Azure).

As-a-service consumption models don't require companies to make large, upfront capital investments. The cloud providers make those investments instead—to the tune of about $68 billion in 2018 from Amazon, Microsoft, and Google combined.[1]

By itself, migrating an application to the cloud doesn't automatically provide agility and scalability. AWS, GCP, and Azure now offer far greater capabilities than they offered a few years ago. To fully capitalize on the latest cloud benefits, companies must rethink their application architectures.

In particular, three fundamental forces are reshaping the landscape and prompting retail and consumer product companies to move quickly and seize advantage:

1. The rise of fast-changing consumer preferences, shifting demographics, and mobile-centric consumer behaviors

2. Increasing competition from both small and nimble startups and forward-leaning large players who use digital and data-driven methods to snag new customers and retain existing ones

3. The always-on, buy-anywhere expectation of today's customer, which has blurred the lines between e-commerce only and brick-and-mortar corporations

[1]Charles Fitzgerald, "Follow the CAPEX: Cloud Table Stakes 2018 Edition," Platformonomics, February 12, 2019, http://www.platformonomics.com/2019/02/follow-the-capex-cloud-table-stakes-2018-edition/.

Cloud native and why it matters

Imagine "lifting and shifting" an on-premise, hosted, three-tier internet e-commerce application to the cloud. The app stack may be fully or partially virtualized. Its code base might run into thousands of lines. Depending on the degree of planned integration and other factors, this migration could take developers and systems professionals a few months to complete.

In our scenario, the migration goes well. Within a few weeks of go-live, the marketing team initiates a series of digital campaigns to drive site traffic. One of three things is likely to happen:

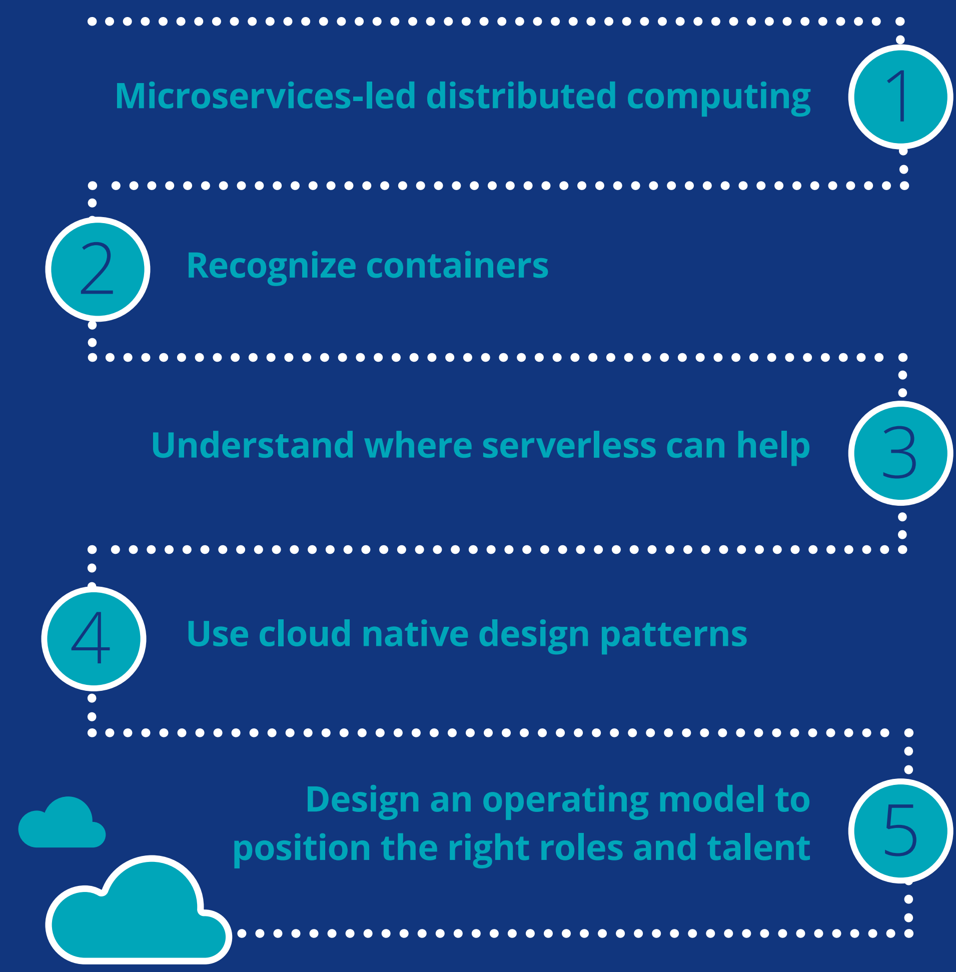
1. The app takes up some light user loads and supports the new traffic. Everything looks great.
2. The app takes a beating, and user access and experience is poor. The system appears to be fine online, but application performance is far below expectations. Shopping cart abandonment rates skyrocket.
3. The app functions well initially but runs into one or more snags, such as checkout failure, site timeouts, or regional issues. Customer complaints pour in.

To fix these issues, the development and production teams will have to grapple with a monolithic application stack with thousands of lines of code. All builds for issues and new functionality will have to feed into a monthly deployment release schedule. New feature development takes a back seat. This scenario doesn't sound like the sunny future the cloud promised.

What could this migration team have done differently? How can an organization engineer its migrating applications to make cloud transition smooth? The answer is to change the playbook from application migration to application modernization—in other words, to go beyond merely using the cloud and instead becoming cloud native. A cloud native approach can make complex applications easier to manage and allow systems personnel to work in more agile ways with quicker release cycles.

The cloud native journey involves five foundational pillars. Each of them is essential to developing a common understanding throughout the organization and to building the right capabilities for the business.

The imperative to become cloud native



1 Microservices-led distributed computing

Traditional application architectures are tightly coupled and monolithic. In contrast, loosely coupled microservices come together to provide distributed computing in the cloud in a way that lets applications remain independently scalable, resilient, and fault-tolerant. The modularity that accompanies these applications allows developers to work in fast, independent teams that focus on specific microservices and can release new code without disrupting other parts of the application.

A microservice focuses on a piece of the transaction that encompasses a unique data boundary. For example, in a typical retail e-commerce application the focus of the "orders" microservice is to register orders from end users. It communicates with other focused microservices, such as the billing service, which processes billing and generates payment receipts for an order.

Following this example, the term "loosely coupled" can imply three aspects:

A. **The orders microservice controls and exposes required functions and data sets through an Application Programming Interface (API).** Other microservices use this API to communicate with the orders service. Additionally, it may use an event-messaging engine to publish transactions. Any microservice "interested" in those transactions subscribes to those topics and may invoke its own transactions based on event triggers. For example, the rewards microservice will generate points for completed orders and tally total rewards from customer activities.

B. **The microservice is independently and horizontally scalable. Based on the number of orders received, the microservice can spin up another instance of itself to handle the traffic.** Or it can scale down to a baseline set of instances with minimum traffic. Thanks to a fault boundary, when one service has an issue, it should not affect other microservices. This approach helps the system minimize impacts to the user experience.

C. **The focus of the microservice is within a single data domain. For example, delivering promotions will focus on promotional data and promotion types as one domain.** Loose coupling presents the challenge of persisting data, so the microservice may choose to persist data locally. Accordingly, unlike a traditional monolithic application stack, transactions that the application undertakes may not be written to a single database. Each microservice may persist through its domain data in a local database. As a result, the system exhibits a combination of strong and eventually consistent data.

2 Recognize containers

Each modern microservice application is packaged and deployed through a container—an isolated and immutable runtime that encapsulates an application. For example, the rise of Docker as a container runtime, or Kubernetes as an orchestration layer for containers, directly benefits microservices. The container keeps deployment efficient by using a small footprint as it shares some of the base binaries and libraries of the underlying operating system. Developers use isolation to develop microservices in their own choice of languages. An application may be built out of two microservices—one written in Java and the other written in Go. This isolation also allows development pipelines to run independently of one another. A change to one microservice can have zero impact to another, which reduces deployment risks and increases the number of releases per application.

The microservice logic and functionality may result in selection of unique components—so, for example, a high-traffic Web front end may choose to persist with some session information in a fast NoSQL Redis database, while another microservice may choose to use a relational MySQL database for slower reads and writes. Yet another microservice may back up these databases into Microsoft Azure Blob Storage service to fulfill data availability requirements.

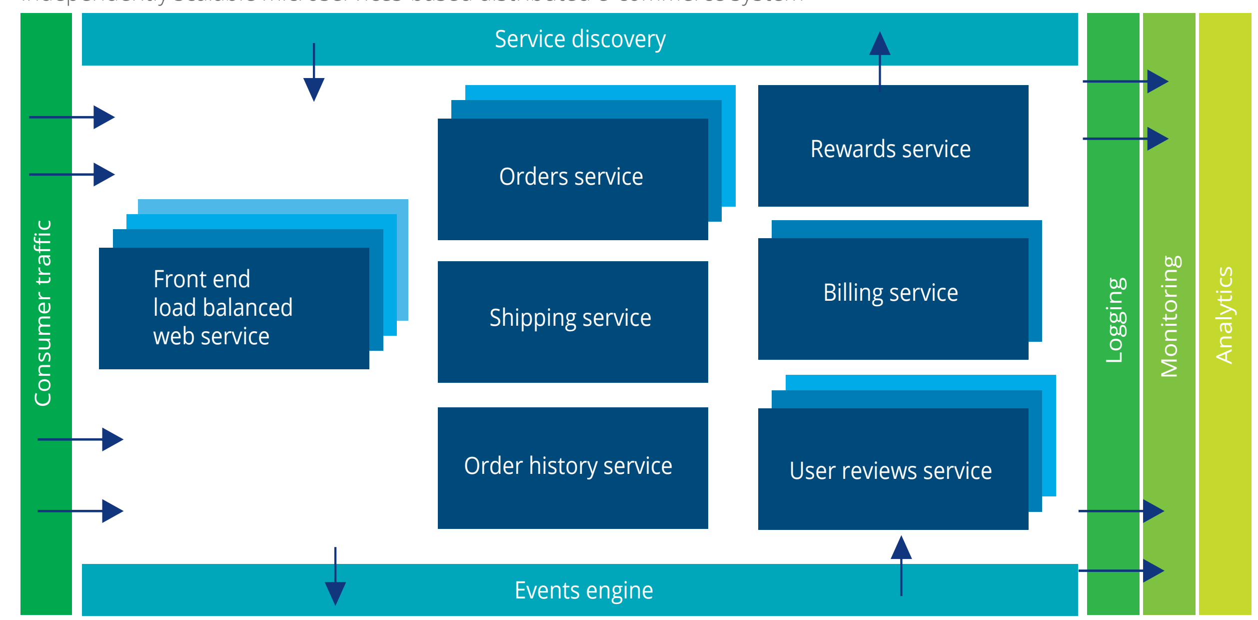
Immutability plays well into the needs of systems professionals: Once deployed, the container runtime is read-only and cannot be changed. This change is a paradigm shift in the way applications are deployed, secured, and maintained in production. To fix a bug or deploy an enhancement, a deployment pipeline will kick off to replace the previously

faulty container with minimal interruptions to operations. Security professionals also like this model because its design thwarts malicious attempts at changing production code.

Because of the ability to use multiple languages, container-based development brings new levels of efficiency to retail and consumer organizations because it permits them to hire professionals with different skills or to contract out parts of the development to organizations with the required talent pools.

Additionally, containers are fast becoming the bedrock of the modern hybrid cloud. Leveraging Kubernetes as an orchestration layer paves the way for portable deployments of containers. This approach brings new levels of utilization for on-premise capital investments in data center hardware while benefiting from the scalability and cost efficiencies of the hyperscale cloud providers.

Independently scalable microservices-based distributed e-commerce system



3 Understand where serverless can help

Serverless architectures offer a compelling way to go all-in on cloud. The Cloud Native Computing Foundation (CNCF) defines serverless computing as the concept of building and running applications that do not require server management. This is not literal—the serverless model just transfers server management programmatically to the cloud hosting provider.

Most importantly, the cloud provider's automation—no humans—is what differentiates serverless architecture from legacy infrastructure outsourcing. Serverless computing deals with a few patterns. One is based on triggers and functions. Also known as Functions as a Service (FaaS), an event trigger from an application can result in a function being called that results in an action. For example, uploading a file to a storage bucket can trigger a serverless function to start the compute instance of an application to begin file processing. When the processing is done, another function may shut down that compute instance. Another pattern may be spin-up and spin-down of several compute resources on the fly in response to processing of a compute-intensive program. Cloud providers have taken advantage of this capability: for example, on the Google Cloud Platform (GCP), the entire portfolio of data analytics and machine learning products is completely serverless.

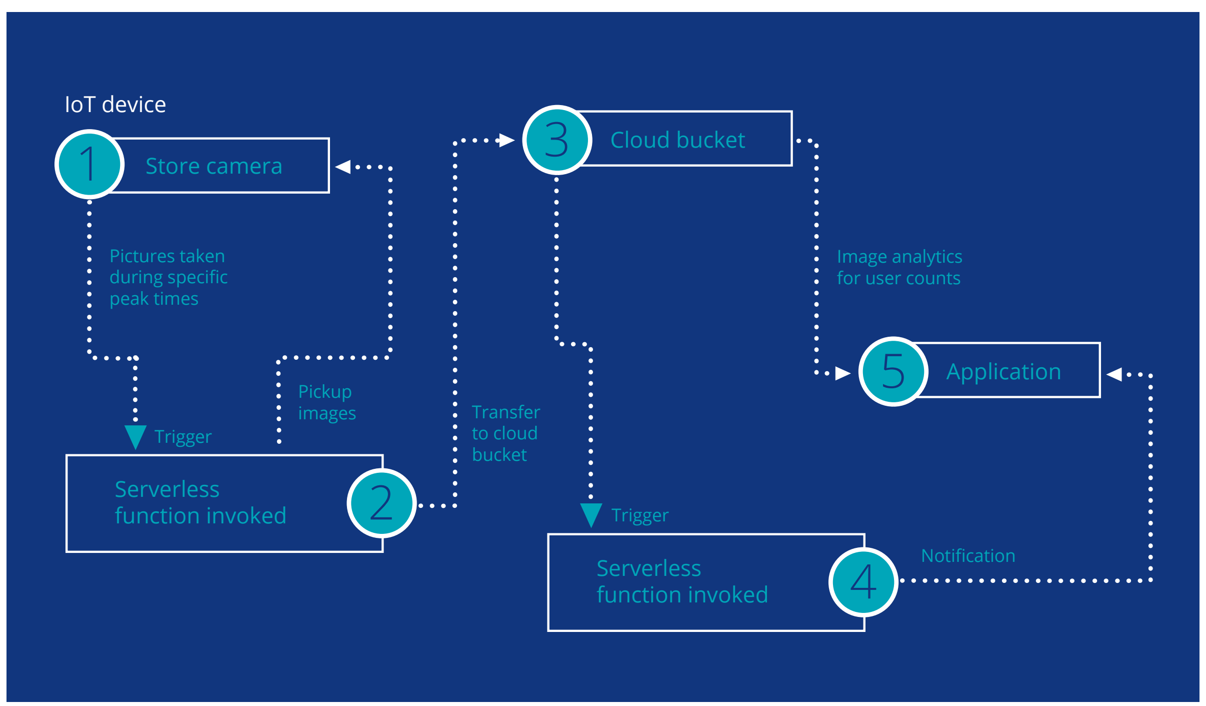
Serverless lets developers focus on code and advances event-based computing. All public cloud providers offer serverless solutions. Additionally, the Apache Software Foundation has highlighted open-source solutions for serverless computing.

Serverless is also gaining popularity for its cost model, given that charges are only applicable for the time a function runs. This approach has ramifications for cloud economics: It allows organizations to pay only for the number of invocations or requests and the associated compute time when the function is running. Remember, virtual machines need to be paid for even when they are idle.

Serverless complements microservices because of the loosely coupled nature of its architecture. While it may not be the silver bullet for every issue, such as long-running jobs, it is a key architecture pattern to consider while developing cloud native solutions.

As an example of serverless, retailers can analyze foot traffic to stores and classify customer gender through attire while masking user images leveraging a machine learning application. This analysis can provide new insights to retailers in terms of gender, time of day and week, and visit versus browse patterns in each store. That information in turn can contribute to increased sales through better positioning of merchandise at points of customer contact.

Serverless functions in action



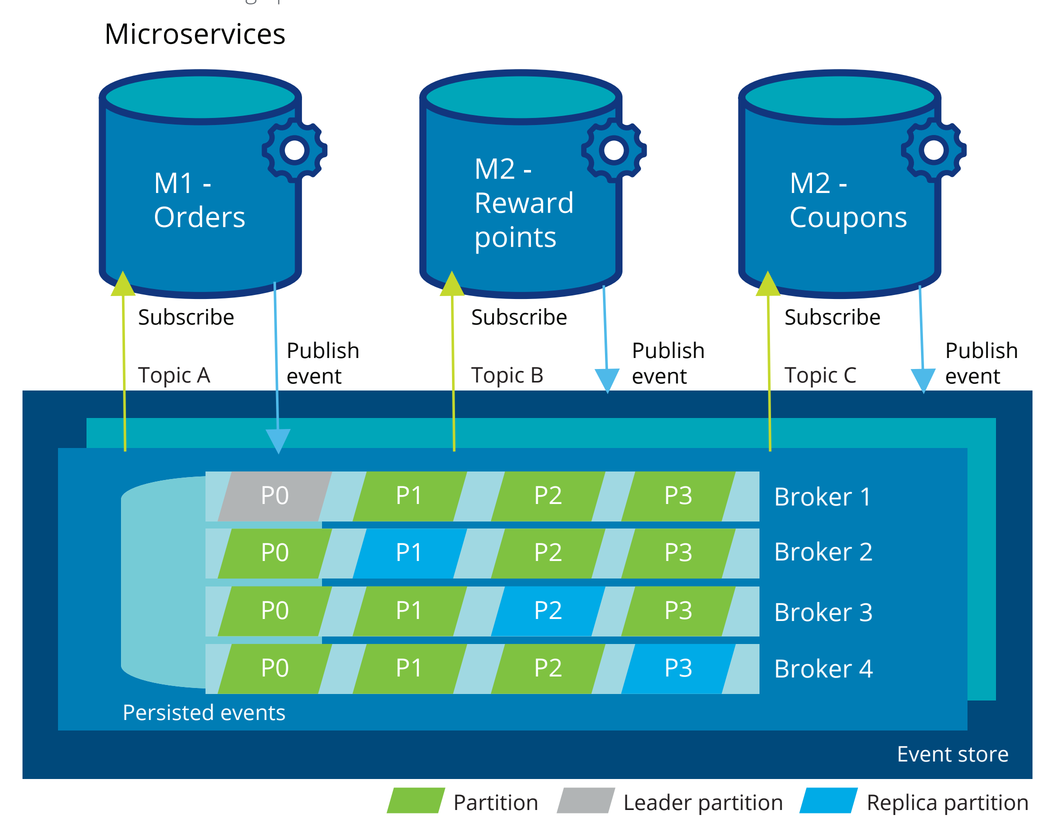
The imperative to become cloud native

4 Use cloud native design patterns

Different cloud native design patterns contribute in different ways to help microservices communicate and interact with each other. One basic pattern is event sourcing. A central event-sourcing engine provides the capability to publish and subscribe to the events, or tiny bits of data, a microservice sends out. So, in the case of an e-commerce example, other design patterns facilitate cloud native architectures through load balancing, auto scaling, and service discovery among related functions. Adopting a set of common standards, reference architectures, guidelines, and design patterns can help product teams, enterprise architects, and systems professionals achieve a common understanding and speak the same language.

Cloud native architectures aren’t all new. Many of the patterns we now call cloud native had precursors in the pre-cloud world. Some of the Web-scale companies have spent several years building and perfecting them. That’s how Netflix can stream a movie to millions of customers simultaneously without hiccups. Public cloud services have only made it easier to expose and absorb these technologies within the enterprise.

An event-sourced design pattern



5 Design an operating model to position the right roles and talent

Going cloud native starts with rethinking traditional IT operations and the structures that underpin teams and productivity. Too many organizations have taken initial cloud native steps only to find themselves stymied by organizational inertia and fixed methods.

Just as microservices stress independence and decoupling, so must the IT organization embrace true DevOps—in which the goal is to be able to build, deploy, and run with much smaller teams that spend less time coordinating with other teams. Organizations need to understand their technology strengths and areas of improvement while designing and executing this new operating model.

Market and external forces make it clear that retail and consumer product companies need to adopt a nimble, product-friendly approach. Cloud native techniques make it possible for them to answer the call. A product team can be responsible for a unique functionality, such as analyzing user behavior, or a set of functionalities aligned to a business, such as promotions.

For that reason, the product team needs to become the core unit of the new organization. It can include architects, developers, and systems professionals who have complete flexibility and independence in building and deploying the product. Amazon's CEO Jeff Bezos famously said that a team that cannot be fed with two pizzas is too big. Heeding this advice can make teams nimble and more effective.

This approach doesn't mean production won't be secured and managed by dedicated and centralized teams. What the product team implies is a focus on velocity and productivity. Cloud native and containerized microservices provide the isolation capabilities teams need to operate with relative autonomy. This capability is one of the key ways some technology companies deploy several releases a day into production, while large traditional enterprises struggle to put out even a few releases every year. This rapid velocity drives small and frequent releases into production—a "test, release, and fix" approach that is distinct from trying to get everything right in one big release. Because being cloud native involves a multitude of technologies, talent hiring needs to be deeper and more involved.

Final thoughts

"Going cloud native" is a paradigm shift in the way an organization can design, build, and deploy technology. Retail and consumer product company leaders can not only reduce operational complexities and costs, but also simplify the IT landscape and drive more frequent production releases. During a time of industry convergence and disruption, retail and consumer product IT organizations need to change from within to help lead the way and help their business counterparts navigate the future. Cloud technologies and cloud native applications represent a big step in the direction toward building a resilient, efficient organization that will last for years to come.

Contacts

Ken Corless

Ken is a principal and the CTO for Deloitte Consulting LLP's Cloud practice and specializes in large transformational IT programs and advising CIO clients on how to handle the disruptive changes going on in the industry.

kcorless@deloitte.com

+1 312 486 3061

Habeeb Dihu

Habeeb is a principal in Deloitte Consulting LLP's Technology Strategy and Architecture practice with a focus on M&A IT within C&IP as well as other industries.

hdihu@deloitte.com

+1 312 486 4970

Ram Ramdattan

Ram is a specialist leader in Deloitte Consulting LLP's Cloud Engineering practice and has served leading clients across retail, financial services, and public sector segments to enhance IT capabilities and improve operating efficiencies.

rramdattan@deloitte.com

+1 469 417 2179

This publication contains general information only and is based on the experiences and research of Deloitte practitioners. Deloitte is not, by means of this publication, rendering business, financial, investment, or other professional advice or services. This publication is not a substitute for such professional advice or services, nor should it be used as a basis for any decision or action that may affect your business. Before making any decision or taking any action that may affect your business, you should consult a qualified professional advisor. Deloitte, its affiliates, and related entities shall not be responsible for any loss sustained by any person who relies on this publication.

As used in this document, "Deloitte" means Deloitte Consulting LLP, a subsidiary of Deloitte LLP. Please see www.deloitte.com/us/about for a detailed description of the legal structure of Deloitte LLP and its subsidiaries. Certain services may not be available to attest clients under the rules and regulations of public accounting.