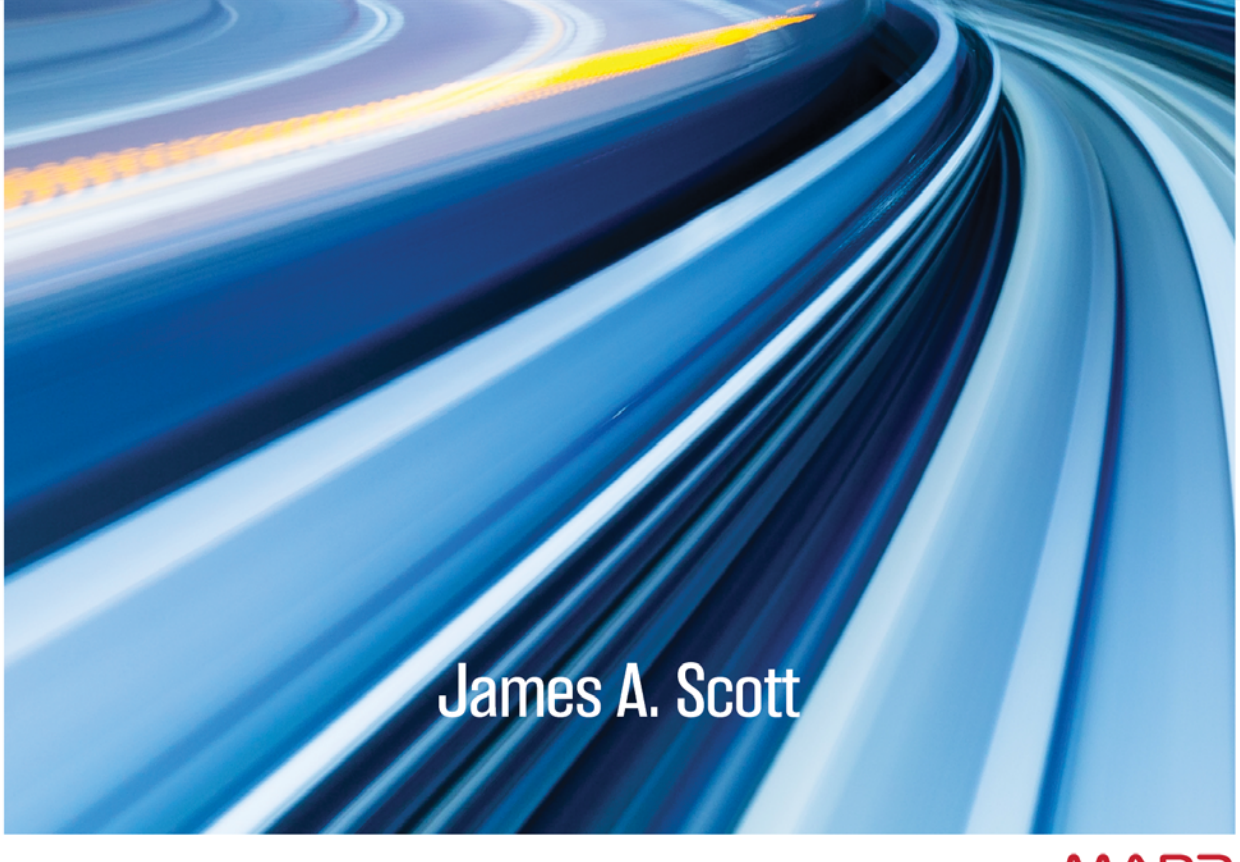
A PRACTICAL GUIDE TO

MICROSERVICES AND CONTAINERS

Mastering the Cloud, Data, and Digital Transformation



James A. Scott

Compliments of MAPR DATA TECHNOLOGIES

A Practical Guide to Microservices and Containers

Mastering the Cloud, Data, and Digital Transformation

James A. Scott

Table of Contents

Introduction

The re-platforming of the enterprise IT infrastructure is no small undertaking, by any means. Perhaps that is why it happens only once every 25-30 years, and usually it is provoked by a shifting set of key business drivers. That is precisely the case today. The term **digital transformation** is in the hearts, minds and on the lips of top-level business executives and IT leaders alike. As we'll see, it refers to entirely new, all-digital ways of doing business and making decisions. And the underlying traditional or legacy infrastructures that have dominated enterprise IT for nearly 30 years simply cannot handle the workloads or power the applications that will drive business decisively forward in the decades ahead. New infrastructure, new thinking and new approaches are in the offing, all driven by the mantra 'transform or die.' The digital on-ramps to transformation originate in emerging highly scalable, very reliable converged infrastructure.

Both microservices and containers are destined to play a major role in this enterprise replatforming, for different reasons. Containers for their part hold significant benefits both for developers and the development effort as well as for the organization itself. Understanding this range of benefits is instrumental to securing funding for containers going forward.

Included in the long list of benefits is the ability to optimize hardware the company already owns, lessening the need to purchase as many new servers. Attracting new IT talent means offering them the technology they want to use, and increasingly that means containers. And most IT staff familiar with Linux and virtual machines should be able to acclimate to containers very easily. Being open source, containers can help reduce acquisition costs and do away with hardware vendor lock-in. Finally, because containers require no particular kind of programming framework or server, developers can write in the language of their choosing, with the flexibility of running them on both Linux and Windows.

For their part, microservices take the decades-old Service Oriented Architecture (SOA) on a leap forward by breaking each SOA component into de facto single purpose applications, but which perform only one function or activity (hence the name microservices). Organizations seeking to make the most of microservice models will increasingly deploy them within or inside containers. This will enable microservices to scale quickly owing to a simpler deployment model. Seen this way, microservices will be a key component of a highly scalable and flexible infrastructure that is straightforward to use, deploy and to maintain.

This book is meant for a wide audience, but in particular targets IT architects; developers and development managers; platform architects; cloud specialists; and big data specialists. For them, the goal is to help create a sense of urgency they can present to their CXOs and others whose buy-in is needed to make essential infrastructure investments along the **journey to digital transformation**. However, some line of business managers as well as upper level IT managers will also benefit from many parts of this book.

The book is further intended both as a highly urgent call-to-action for IT, as well as an overview of the most promising and compelling tools, technologies and solutions destined to play a major role in this re-platforming of the enterprise. The book is designed to help you stay one clear step ahead of your peers and competitors as IT undertakes what is arguably its most mission-critical task in a generation. Nothing less than the survival and competitive capability of your organization is at stake.

What is Digital Transformation and Why the Urgency Today? 1

There are both consistent and compelling reasons why digital transformation is in many cases the hottest topic in boardrooms, conference rooms, and around corporate lunch tables. Business and IT leaders alike have come to the common conclusion that they must transform the way the enterprise does business or risk going out of business entirely. In the recent Voice of the Enterprise Storage survey from 451 Research of 500 senior IT decision makers, two thirds of respondents said their businesses will require moderate to significant transformation in the next five years.[1]

For example, venerable banks must transform to deal with the literally hundreds of well-funded 'fin techs' – non-traditional, well-funded, low-cost online-only banks gaining great favor with millennials and other high-growth demographic groups. Another venerable industry, hospitality, is being rocked to its core by the likes of Airbnb, HomeAway and others and their all-digital approach to linking global travelers to rooms. Businesses stood in awe as Uber and Lyft utterly and irreversibly disrupted a century-old industry by leveraging their digital native expertise and the ubiquity of smartphones.

Power of digital transformation

Digital transformation is essentially a remaking and reforming of how an enterprise serves all its constituencies, including its customers, employees, and business partners. It further defines processes that support continuous operations improvement, fearlessly disrupting existing businesses and entire markets while inventing new business models and even new businesses along the way. And as the name implies, digital transformation fully leverages digital technologies in a highly strategic, carefully planned way to effect these profound changes. Just consider the examples above as illustrations of the awesome power of digital transformation.

Thus it is business imperatives that are driving digital transformation. The question then becomes what will enable digital transformation?

1 451 Research, Voice of the Enterprise Storage survey of 501 senior IT decision makers, Q2 2016, **https://451research.com/dashboard/customer-insight/voice-of-the-enterprise/voice-of-the-enterprise_storage**

The answer is, nothing short of the most significant overhaul and re-platforming of the IT infrastructure in the last 30 years (see fig.1-1 below). And the reason is this: The fast-emerging newer technologies that will drive the applications and workloads in the transformed enterprise include the likes of **big data analytics**, containers, Internet of Things (IoT), and hyper-aggressive cloud adoption. The traditional or legacy systems initially put in service in the late 1980s simply cannot support these technologies in an efficient or effective way, if at all. This infrastructure overhaul will be the key on-ramp to **digital transformation**.

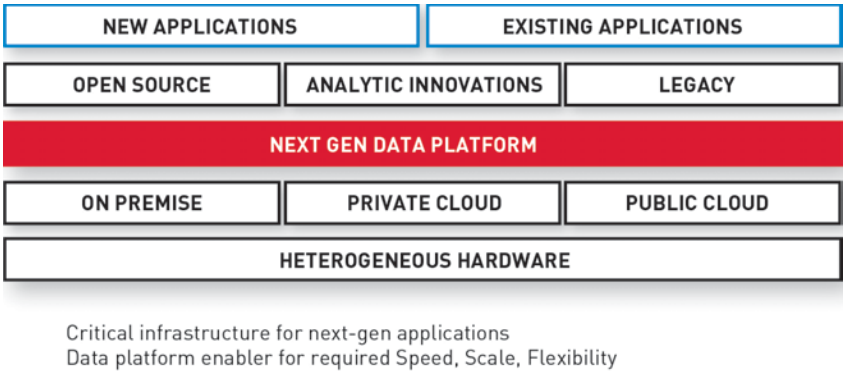


FIGURE 1-1

Fully leveraging all the data

As an example consider one major technology for which those legacy systems were built, namely data warehouses. Using archived data that was scrubbed, structured and otherwise made 'warehouse-ready', IT would apply various business intelligence reporting tools to view this historical data and mine insights to drive decision-making. This whole process took time, weeks in some cases.

Today, businesses compete ferociously to boost time to market of mission-critical products and services. Doing so means working with ever-larger data sets from an expanding universe of data sources, including social media, third parties, government sources, private research sources, IoT sources, and of course a constant in-flow of operational data. Increasingly this data is semi or completely unstructured, and legacy systems choke on both data types.

All of this is incompatible with the desire of businesses to marry operational data in real time to the output of business analytics tools from data warehouses. (See fig. 1-2 below). It isn't enough to simply have this data warehouse output analyzing sales and other data from the previous quarter. Rather businesses need to match those insights with real-time

analysis of current operational and other data to gain entirely new insights and make far more timely business decisions.

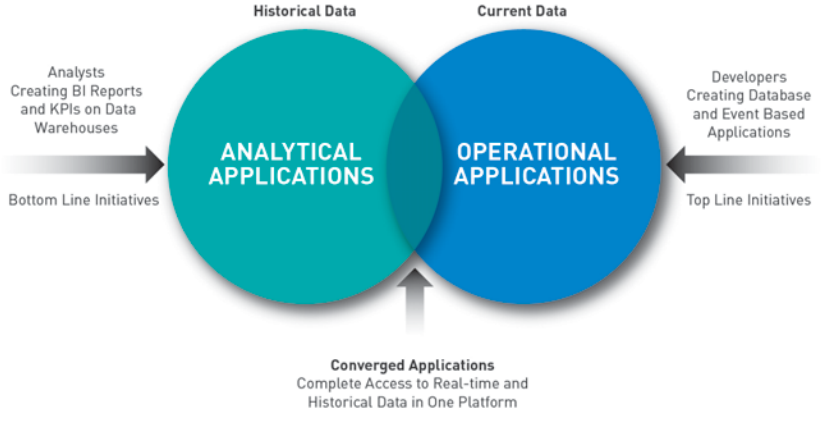


FIGURE 1-2

Data siloes need not apply

But this cannot be done when warehouse data resides in one cluster and operational data is in another. Something new is needed, and that is converged data platforms, which will be discussed more fully later on. Think of converged infrastructure as the planned, strategic combination as well as integration of all computing resources – hardware, networking and software – effectively eliminating silos and allowing the seamless yet secure flow of data between clusters.

Enterprises are already at work shifting spending to infrastructure that will support digital transformation. As shown in Fig. 1-3 below, over the next four years, companies will experience flat IT spending. But underneath that will be a steady decrease in legacy spending accompanied by a corresponding increase in spending behind next generation technologies. The key to reducing costs while driving innovation is the data. In fact, the forecast also shows that within four years 90% of data and workloads will be handled on next generation technology.

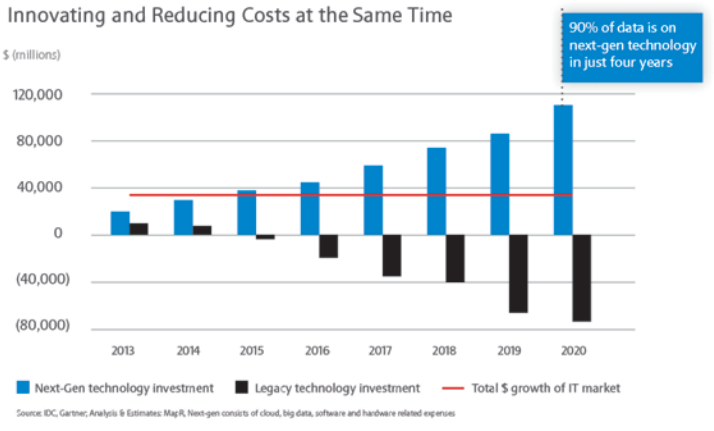


FIGURE 1-3

The bottom line is that enterprises that experience the most success expanding market share will be those not with the most data but with the most data agility, which is the ability to generate the fastest and most appropriate response to changes in customer demand.

2 Typical Problems Facing Enterprise IT on the Road to Digital Transformation

This chapter provides an overview of the many challenges facing organizations today as they initiate or continue their **journey to digital transformation**, as well as the vital re-platforming of the IT infrastructure. While the focus here is on problems and challenges, specific solutions to these problems will be the chief focus of most of the rest of this book. Whatever specific problems you and your organization may be facing, rest assured you are not alone. These once-in-a-generation infrastructure overhauls are never easy. Not only does IT management have to deal with traditional and legacy infrastructure that is inadequate for the very high scale and low latency requirements of emerging technologies, but it must also come to terms with legacy 'thinking' as well as legacy skillsets. It is a multi-faceted challenge, but one that must be overcome for the sake of the long-term viability of the enterprise.

Three emerging technologies merit particular attention here as current challenges, the resolution of which shall be addressed later on. They are:

- Cloud computing
- Big data and big data analytics
- Containers

Cloudy cloud

The data on the global movement of workloads to various cloud environments is as unambiguous as it is compelling. According to a survey of more than 900 senior IT decision makers by 451 Research[1], nearly 59% of enterprise workloads that were placed in non-cloud environments in 2016 will shrink dramatically to 40.5% by 2018, representing a stunning 31% drop. Over this time workload deployments to SaaS environments will jump 63%;

1 451 Research Voice of the Enterprise Cloud survey, Q2 2016, **https://451research.com/dashboard/customer-insight/voice-of-the-enterprise/voice-of-the-enterprise_cloud**

deployments to hosted private clouds will rise 41%; and deployments to IaaS environments will more than double at 112%.

Perhaps the biggest challenge facing IT with respect to cloud is the obvious realization that there is no one single cloud. Rather there are many, and most enterprises will deploy applications to several of them simultaneously. The challenge is one of orchestration and integration of data across various clouds.

For example, consider the app/dev environment. Ideally a lot of prototyping and testing would be done in the public cloud, with its scale-on-demand capabilities that make it easy for developers to get this vital work done without imposing on internal resources. Most organizations still choose to retain their most sensitive data on premises, however. Yet to complete the app/dev process from development to test to production, data must flow securely and seamlessly among these different environments in what is known as the hybrid IT world (a combination of on-premises and off-premises public and private cloud as well as use of traditional on-premises non-cloud clusters).

What is sorely needed to make all this happen is a distributed processing model that scales easily across all locations and environments. In other words, a converged data platform.

Big Data and Big Data Analytics

One of the biggest challenges organizations have today with big data, which includes IoT data, is that it is increasingly of the semi and unstructured variety. Legacy RDBMS simply cannot aggregate, store and process this data efficiently or effectively, certainly not in high volumes. The data volumes are unprecedented and growing at warp speed, in some cases doubling annually by some estimates. IT is under tremendous pressure to deal with the volume, variety, and velocity of new data, while at the same time pressured to deliver more personalization and better service to the customer base. This complexity of big data environments is illustrated well in Fig. 2-1 below.

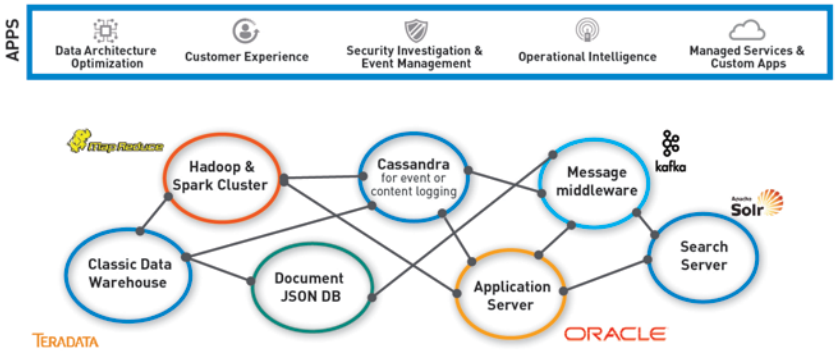


FIGURE 2-1

This environment itself is highly dynamic, with frequent new releases of the many different components that make up the big data world. Other challenges abound:

- Storing all the data is difficult, with many organizations looking to **data lakes** to collect and store huge volumes of data in native formats. But constructing them properly can be tricky, at times resulting in repositories where data is collected and seldom retrieved.
- **Big data analytics skills** do not grow on trees! Shortages of so-called data scientists have been well documented, and shortages of people with critical skills translates into very high salaries to acquire them.
- According to a recent major global security report, security worries easily top the list of barriers to more aggressive deployments of big data platforms.
- Apart from the storage issue, there is the big data conundrum of having systems capable of managing these mega-data streams from so many disparate sources. From within the enterprise the sources frequently cut across multiple departments. And the number of potential outside sources is growing rapidly, including all of social media, third party data, government data, reports and research from the scientific and academic communities, industry data, and on and on.
- Getting effective ROI from big data efforts are closely related to how quickly you get the results from the big data analytics tools. This process needs to happen at the speed of business, but in many organizations it does not, owing to all the complexities of the big data environment outlined here.

Containers

As will be addressed more fully in Chapter 4, containers are one of the fastest growing new technologies, owing to their capability to usher in a new and vastly improved application development environment. You can think of containers as operating system virtualization in which workloads share operating systems (OS) resources. Though they have been around for just a few years, their adoption rate and acceptance is impressive to say the least. In one major global study[2] of 1100 senior IT and business executives, 40% of respondents said they are already using **containers in production**, half of those with mission-critical workloads. Only 13% say they have no current plans to use containers this year, with the remainder all making container usage plans.

Containers can do a lot of what virtual machines (VMs) cannot do in a development environment. They can be launched or abandoned in real time instantly, which VMs cannot. Unlike with VMs, there is no requirement for OS overhead in the container environment. And containers are destined to play a major role in facilitating the seamless transfer of development from one environment or platform to another.

However, there are issues and challenges with containers as well. For one, most containerized applications today are stateless. This could be an issue for stateful applications, but as will be shown, there are workarounds. These workarounds include solutions that provide the reliable storage needed to support stateful applications.

As with big data platforms, containers do provoke data security concerns, but this is not unusual with relatively new technologies deployed in critical areas such as application development.

2 Thales 2017 Global Data Threat Report of 1,500 organizations in seven nations. **https://www.thalesgroup.com/sites/default/files/asset/document/thales_2017_data_threat_report-global_edition.pdf**

3 In Search of Agility

This chapter will focus on the concept of agility as it relates to digital transformation. The various aspects of agility – namely data agility, application agility, and infrastructure agility – will be examined in far greater detail in the following chapters. The aim here is to separate agility from the many other buzzwords that flood the IT and business worlds, and demonstrate the intimate link between agility, digital transformation, and enterprise success. As noted in the introduction, digital transformation is a phrase frequently used throughout the organization today by managers of all stripes. And, as noted, converged infrastructure is a key on-ramp to digital transformation. The most coveted result or 'output' of this newly forming infrastructure is agility, which also happens to be one of the more overused terms in both the IT and business suites. Stripped of all else, agility describes how quickly an enterprise can respond to new opportunities and new threats.

The Innovator's Tips and Tricks, by Quantium
Benefits derived from putting all data in one linearly scalable platform include:

- Elimination of data silos
- Easier scalability
- Enhanced tool-sharing across the organization
- Ability to accommodate multiple workload types, such as batch, real-time and ad hoc reporting

In a recent major cloud study[1], respondents were asked why cloud solutions were used on a variety of workloads, including email, analytics, big data, application development, and several others. For virtually every workload the top one or two reasons selected were

1 Research Voice of the Enterprise Cloud survey, Q2 2016, **https://451research.com/dashboard/customer-insight/voice-of-the-enterprise/voice-of-the-enterprise_cloud**

'responding faster to changing business needs.' In other words, organizations are seeking greater agility from cloud as well as other advanced technologies including big data analytics and containers. Cost savings, which for several years was the top justification offered to higher ups for cloud investments, is fading in importance as C-level executives grasp the business value of agility.

Data agility

Organizations have traditionally been hamstrung in their use of data by incompatible formats, rigid database limitations and the inability to flexibly combine data from multiple sources. Users who needed a new report would submit requirements to the IT organization, which would place them in a queue where they might sit for a month or more. Even worse is that users have had to know in advance precisely what data was needed. Ad hoc queries were only permitted within the confines of an extract database, which often contained incomplete and outdated information. Queries were limited to structured data. Data agility encompasses several components:

- Business users are freed from rigidity and given the freedom to combine data from multiple sources in an ad hoc manner without long cleansing or preparation times.
- The path between inquiries and answers is shortened so that decisions can be made on current data.
- Structured and unstructured data can be combined in meaningful ways without extensive transformation procedures.
- Data can be combined from both operational and analytical (historic) sources to enable immediate comparisons and to highlight anomalies.
- Data can be combined from both streaming and static data sources in real time. Users can create their own integrations using visual programming tools without relying on time-consuming extract/transform/load procedures.
- New data sources can be quickly integrated into existing analytical models. Schemaless data is supported in flexible formats like JSON.
- Support for combinations of complex structures such as JSON documents with simple key-value constructs and tabular formats.
- Block-level, file-level and object data can be combined in the same model
- Rich visualization tools enable business users to create graphical representations of data that reveal trends and relationships that would be otherwise hidden.
- Instead of specifying which data they need, users can access all available data for experimentation and discovery.

- Users can create and share their own analytical models without disturbing production data.

In a nutshell, data agility is about removing barriers to data usage. The rigid structure of yesterday's data warehouses made data a precious asset that could cost upwards of $10,000 per terabyte. With **Hadoop**, those costs fall by more than 90%. This removes many of the cost and technical barriers of enabling data agility. The most formidable barriers to data agility at many organizations aren't technical, but rather cultural. Functional managers may jealously guard data within their groups, believing it to be a source of power, or the IT organization may see itself as data stewards and tightly limit access. Appropriate protections should always be applied to sensitive data, of course, but the difference between agility and rigidity often comes down to the organization's willingness to trust its people to use data responsibly and strategically.

Another example of data agility is given by a major Europe-based telecommunications giant, which similarly collects veritable mountains of data from its far-flung network operations. The mountains are not important in and of themselves. Rather it is the mother lode of information locked within them, as forward thinking organizations have come to realize. According to an IT manager there, "Our applications allow mobile operators to proactively monitor and improve customer experience. We collect data from the mobile network, then process the collected data to come up with information about individual subscriber experience with mobile services like video usage, web browsing, file transfer, etc. We also create individual subscriber profiles from this data. All of the data is stored in our **MapR Converged Data Platform**. We want to enable mobile operators to do interactive ad-hoc analysis and build reports using BI tools. This is where **Apache Drill** comes into the picture. We're using Drill to enable interactive ad-hoc analysis using BI tools like Tableau and Spotfire on the data we produce. Currently we're building canned reports using Drill to show how the data we produce can be used to derive insights."

Application agility

The containers section in chapter 2 above and the fuller treatment of containers in chapter 4 offer a front row seat to application development agility, which also explains the very rapid adoption of and enthusiasm for containers. In more fully appreciating the value of application agility, it is important first to understand just how completely the application development environment is changing.

As shown in Fig. 3-1 below, this environment has become far more complex and multifaceted in the last decade. The complexity has arisen chiefly due to the great difficulties in dealing with separate clusters or silos of data. This reality in some organizations has made application agility almost a misnomer. Clearly, a way is needed to seamlessly move data from one environment to another without having to customize the data for each environment. In fact, it is widely believed that within a few years developers leveraging containers will be able to do exactly that – move their test/dev projects into production without need-

ing any major code rewrites to accommodate new data locations, such as on-premises or onto a cloud.

Enterprise applications have traditionally been built with a monolithic, vertically integrated structure. All application logic was contained within a single program, which often took months or even years to develop. Detailed functional specs were required and extensive testing and code reviews were part of the process. The resulting application may have fit the stated requirements, but there was little latitude for embellishment. Enhancing an application required the preparation of new functional specifications, extensive testing and more code reviews. Even modest enhancement requests could consume months. The legacy applications used by airlines, banks and credit card processors are typical of these very large, robust, but inflexible programs.

The rapidly changing business and technology landscape of today no longer tolerates this approach. Year-long development schedules create applications that are often irrelevant by the time they are complete. New technologies like bots, automated assistants and mobile devices must be accommodated quickly. New payment mechanisms such as bitcoin and Ethereum come seemingly out of nowhere to change the way customers conduct transactions. Organizations that want to expose data or services to business partners have no way to do so because those needs weren't anticipated when the application was built.

Application agility reimagines development and deployment around a modular, loosely coupled and **dynamic architecture**. Like Lego blocks, the components of the application can be moved and reassembled as needed. Communication between components is provided by a messaging plane. New or enhanced functionality is provided by revising individual components and slipstreaming them back into the network without disrupting the application as a whole. These components, or services, can also be stored in a library and shared by multiple applications. Services are called only when needed, which reduces program size and complexity while improving performance.

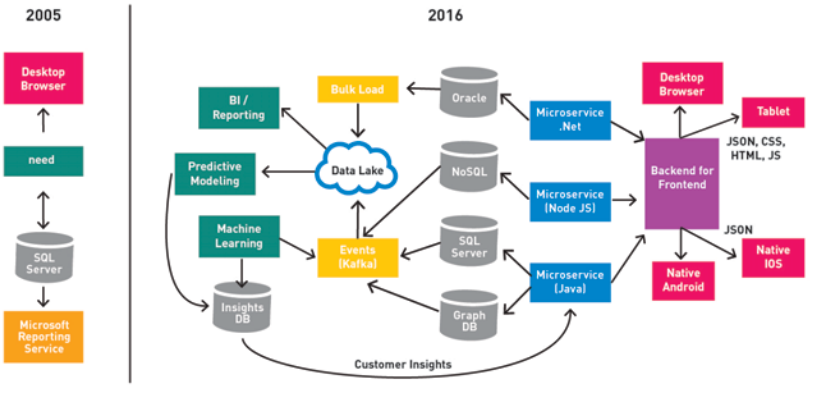


FIGURE 3-1

Application agility also changes the nature of the data that applications process. Less and less, it is data warehouse or data mart data finding its way into leading edge applications. The growth area is event-based data (Fig. 3-2). Whether it's collecting machine sensors to predict and prevent failures, or providing key offers to customers, or identifying and preventing fraud before it happens – all such **use cases** are enabled by event based data flows and a converged platform.

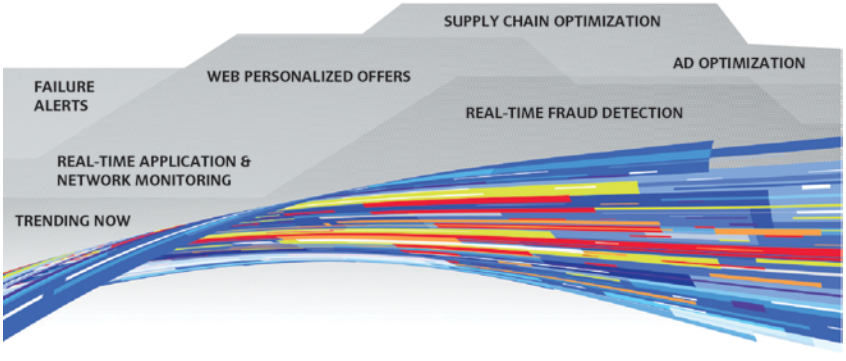


FIGURE 3-2

Infrastructure agility

As we noted earlier, converged infrastructure is the key on-ramp to digital transformation. The once-in-a-generation replatforming currently underway is synonymous with infrastructure agility, because enabling the next-gen applications requires a next-gen infrastructure.

IT infrastructure is undergoing a transformation that is no less radical than that being seen in data and applications. Virtualization has brought unprecedented flexibility to resource provisioning, a **foundation that containers build upon**. Software-defined everything is close to becoming a reality. In the future, infrastructure components such as storage, networks, security and even desktops will be defined in software, enabling resources to be quickly reconfigured and re-allocated according to capacity needs.

Agile infrastructure is also highly automated. Processes that once took weeks, such as the introduction of a new storage controller or network switch, can be reduced to minutes with minimal operator intervention. Policy-based automation predicts and adjusts resource requirements automatically. A data-intensive process requiring rapid response can be automatically provisioned with flash storage, while a batch process uses lower-cost spinning disk.

This kind of agility will be essential to achieving comparable nimbleness in applications and data. Users shouldn't have to care whether software is running on a disk or flash. Networks should automatically re-provision according to capacity needs, so that a videoconference doesn't break down for lack of bandwidth. Storage will be available in whatever quantity is needed. Containers will spin up fully configured with required services.

Most importantly, distinctions between on-premises and cloud infrastructure will fade. Open standards and on-premises mirrors of cloud environments such as Microsoft Azure Stack and Oracle Cloud on Customer are among the forces moving toward complete cloud transparency. Developers will be able to build on-premises and deploy in the cloud, or vice versa. Users should expect workloads to shift back and forth between environments without their knowledge or intervention. Infrastructure agility is effectively infrastructure transparency.

The Innovator's Tips and Tricks by Quantium
When moving to a modern data architecture:

- Start small, pick a workload that has demonstrable business value and see the project through to production.
- Involve everyone. Set milestones and talk about what you learn at each stage

This kind of flexible, self provisioning infrastructure will be required to support big data and analytics. In that scenario, agile infrastructure includes the following five foundational principles:

1. Massive, multi-temp, reliable, global storage with a single global namespace.
2. High-scale, asynchronous, occasionally connected, global streaming data layer that is persistent.
3. Support for multiple techniques of analytics or compute engines.
4. Ability to operationalize whatever happens; operational applications combined in the same platform.
5. Utility grade cloud architecture with DR, workload management, scale-out.

Seen this way, the next generation infrastructure is not an incremental improvement of existing approaches. It truly is a radical replatforming able to bridge new modern applications with legacy systems.

Big data platforms are changing the way we manage data. Legacy systems often require throwing away older data, making tradeoffs about which data to maintain, moving large

data sets from one silo to another, or spending exorbitant amounts to handle growth. But those are becoming the modus operandi of the past. Scale, speed and agility are front and center with the **modern data architectures** that are designed for big data. Data integrity, security and reliability remain critical goals as well. The notion of a 'converged application' represents the next generation of business applications for today and the future.

4 Application Agility

Digital transformation is built upon a foundation of flexible applications and infrastructure. This requires a different approach to building applications versus those that have been used in the past, one that is driven by the need for agility. Traditional, monolithic applications don't lend themselves to agility. Agile applications may be used in many contexts. They typically perform a limited number of functions that can be called via APIs. A simple example of an agile application is a newsletter sign-up form or payment widget which can be embedded in a website. The application performs a single function, but may be used in millions of ways in millions of places.

Agile applications are enabled by virtualization, which enables them to be launched quickly and shut down quickly when no longer needed. Two important infrastructure elements of agile applications are microservices and containers – we'll discuss each here.

Microservices

A **microservices architecture** is a method of developing applications as a network of independently deployable, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal. Think of it like a honeycomb. Each cell in a honeycomb is independent from all the others and may be used for a different purpose. Each single cell isn't very useful, but when combined with each other, a strong and flexible network is created that supports many uses.

There's nothing new about this concept. The vision of a service oriented architecture (SOA) was first sketched out in the 1980s as a way to unify monolithic islands of automation into a functional goal. It's no coincidence that the concept of SOA arrived at the same time as the internet made large-scale peer-to-peer networking possible.

Applications based upon services need other services to manage them. The concept of middleware is based upon this idea. Middleware coordinates groups of services to ensure that data flows smoothly between them and that services are available when needed. The enterprise service bus (ESB) is another way to coordinate services. This is an integration architecture that uses a common communications layer (the bus) to orchestrate a variety of point-to-point connections between providers and consumers. For example, an ESB may

call upon separate services for a shopping cart, a credit approval process and a customer account to present a unified checkout window to an online shopper. In most cases, a common data storage layer is shared by all services.

The difference between a microservices approach and an ESB is that microservices architectures have no single coordinating layer. Each service communicates independently with the others. This enables applications to be assembled quickly and flexibly by teams working in separate functions. They can each be written in different languages, which gives developers flexibility to match the language to the task. Each service can be developed using the programming language most appropriate to the task. The goal in microservices development is to deconstruct the application into the smallest possible parts so that services can be shared and combined easily with other applications.

The Innovator's Tips and Tricks, by Quantium

Some of the gotchas of rolling out and managing a microservices architecture include the following:

- Open source platforms that aren't fully mature may contain bugs that you will have to fix yourself. Be willing to dig into the source code.
- To maximize container security, ensure that access is restricted to administrators only.
- Regularly review containers for security holes before moving into production.
- There are lots of moving pieces in microservices – and container-based environments. That means monitoring is not only a critical success factor bus is a mandated requirement.
- Run a minimum of three to five masters to ensure high resiliency in the face of server outages. The more masters you have, the less pressure is involved in resolving master hardware issues.
- Continually educate business users on the benefits of a microservices architecture. Use a hands-on approach so users have something tangible to work with.

There are several robust frameworks that developers can use to build microservices-based applications. Frameworks incorporate libraries of essential services that developers need to build microservices-based applications. Here are some examples.

- **Spring Boot** is a highly regarded framework for dependency-injection which is a technique for building highly decoupled systems. It's known for simplicity, flexibility

and support for distributed programming techniques like inversion of control and aspect-oriented programming. It also permits developers to choose between multiple web servers like Tomcat, Jetty and Undertow.

- **Jersey** is a RESTful Web Services framework for development of RESTful webservices in Java. RESTful refers to a popular development technique in which messages between microservices are handled with the web-standard HTTP protocol. Known for its ease-of-use, Jersey provides support for JAX-RS, a Java application program interface (API) specification for the development of Web services.
- **Swagger** is a framework of development using APIs. it enables consistent descriptions of APIs that machines can read and that can serve as documentation. Swagger also automatically generate client libraries for API in a wide variety of languages.

The microservices approach to application development has been enabled by a number of new technology and development techniques:

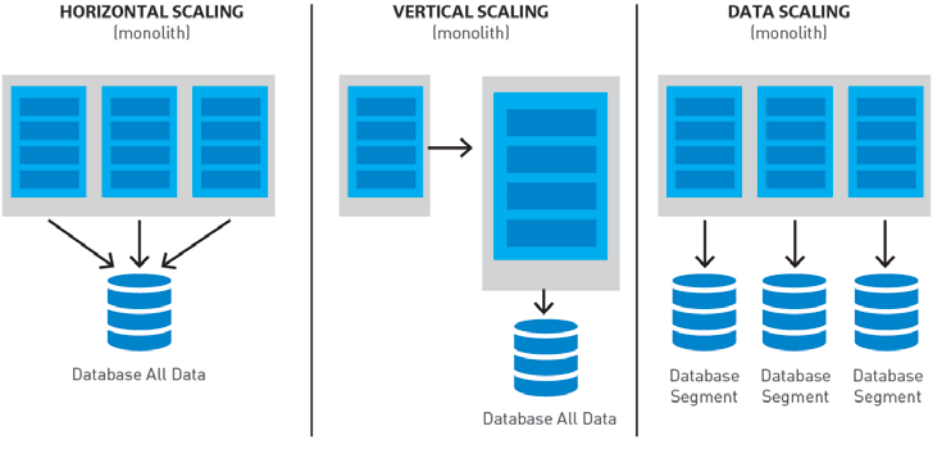
- High-speed, low-latency networks enable sophisticated distributed applications to be constructed of services from many providers. For example, an application can call a secure document signing service or fraud detection service in milliseconds in order to close a sale.
- Containers are lightweight virtual machines that contain only the infrastructure elements necessary to perform a service. They can be launched and shut down quickly with minimal management overhead. Each service can be encapsulated in its own container and stored in a library.
- RESTful APIs are defined by whatis.com as application program interfaces (API) that use HTTP requests to GET, PUT, POST and DELETE data. They're a low-bandwidth way for services to communicate with each other using a standard set of commands. This makes them well-suited to loosely coupled applications on the internet. The population of services exposed as APIs is exploding. ProgrammableWeb lists more than 17,000 APIs, up from just 300 a decade ago. Not all microservices are RESTful, but all are message-driven.
- Distributed databases have multiple services handling data requests. Each service works on a subset of the data and coordinates results with other services via an orchestration platform. This allows for highly scalable applications to be built at low cost using commodity servers.
- DevOps is an agile programming technique that emphasizes modularity, frequent releases and a constant feedback cycle.
- DataOps combines the concepts provided by DevOps and also layers in the management and availability of data models. This enables the ability to quickly productionize intelligent machine learning models without the old approach of throwing the

model over the wall with fingers crossed that someone else will figure out how to put it into production.

An important element of the microservices concept is that services communicate with only a few other services using lightweight, flexible protocols. This might entail using a remote procedure call (RPC) protocol such as REST or a messaging system such as **Apache Kafka** or **MapR-ES**. For developers who are steeped in the techniques of building monolithic applications, microservices requires a different way of thinking. It requires thinking of programs as flows rather than states. The power of this model is its flexibility and scalability.

Microservices and Big Data

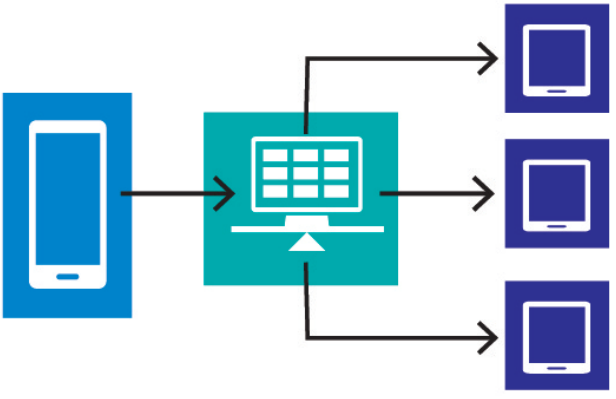
It's no coincidence that microservices and big data have gained popularity at the same time. Both employ similar approaches to managing data. The big data approach, as embodied in Hadoop, partitions data into smaller subsets that are processed close to the data with the batches aggregated into a single result. This creates a highly scalable architecture using low cost hardware.



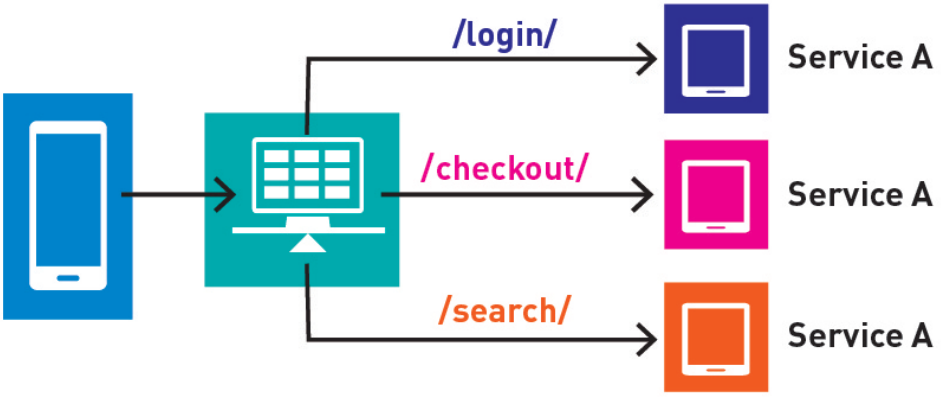
Microservices scale in much the same way. Their modularity supports independent updates/deployments and helps to avoid single points of failure, which can help prevent large-scale outages. There are three basic approaches to scalability[1]:

X-axis scaling runs multiple copies of an application on a shared data set. Workloads are managed by a load balancer. This approach is relatively simple, but it requires more cached memory and does not scale well with complex applications.

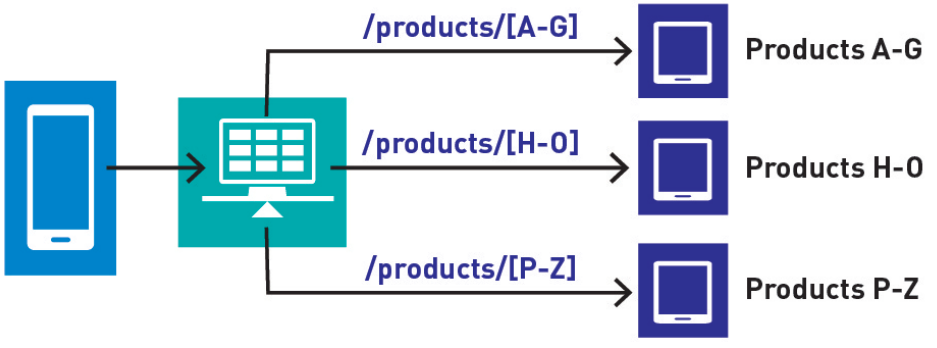
1 Images courtesy of DevCentral

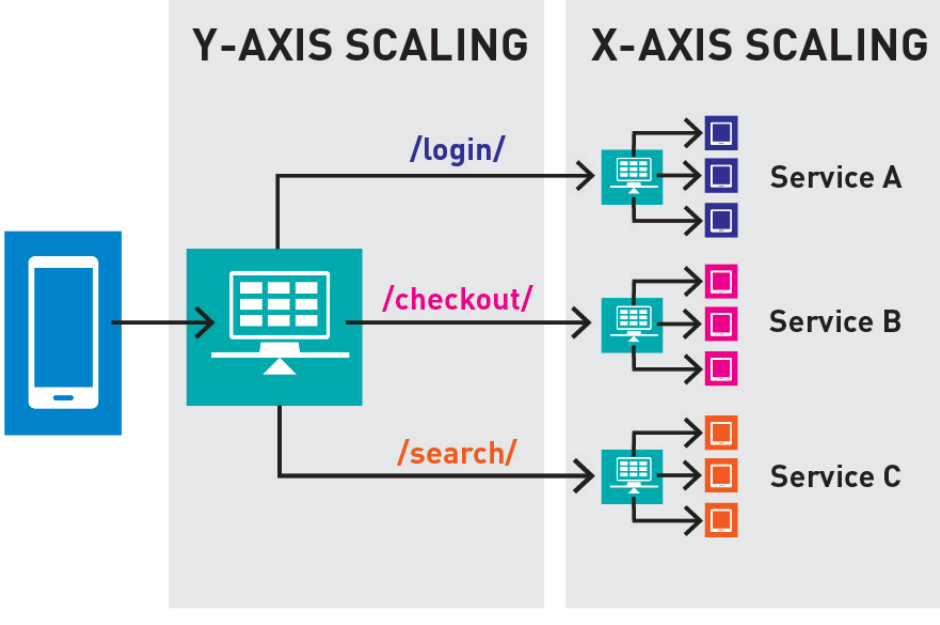


Y-axis scaling splits the application into multiple services, each of which is responsible for a specific function. For example, one service might handle checkouts while another manages customer data. The results are harmonized at the application level. This approach lends itself well to complex applications.



Z-axis scaling is similar to x-axis scaling in that each server runs an identical copy of the code, but in this case each server manages only a subset of the data with a primary key used to partition the routes. A router sends content to the appropriate partition and a query aggregator combines the results. This approach is more memory-efficient and scalable in transaction scenarios, but it has some of the same complexity drawbacks as x-axis scaling.





Microservices are well-suited for modern application development if the design of the application is structured to use services from the ground up. However, microservices can also be called from legacy applications to support new functionality.

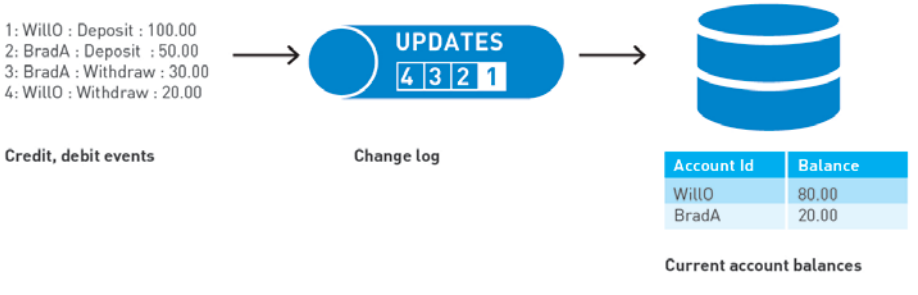
Microservices Design Architecture Patterns and Examples

Event-driven (trigger based) microservices provide a structured way to coordinate multiple services in a highly scalable manner by using message queues and parallel process-

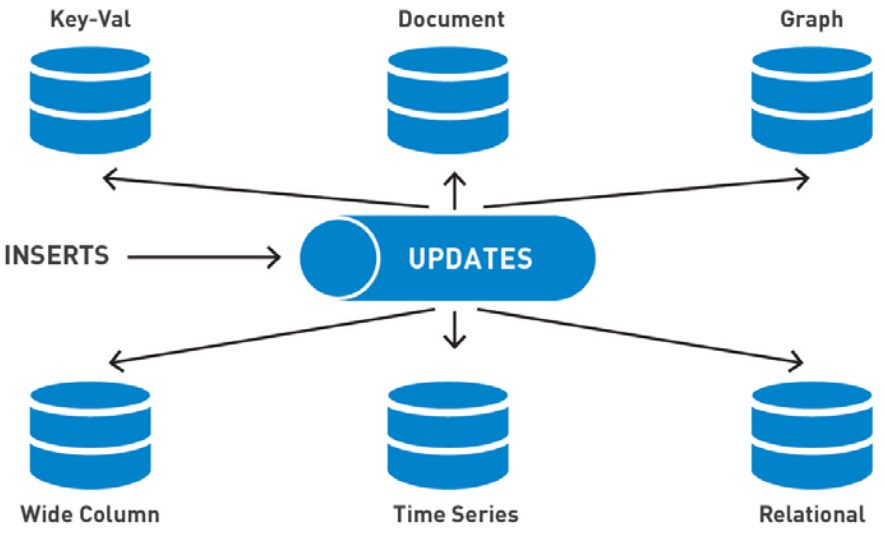
ing. Some common deployment patterns for microservices include event streaming, event sourcing, polyglot persistence and command query responsibility separation. We'll look at each in brief.

Event streams leverage **streaming engines** such as **Apache Kafka** or **MapR-ES** to capture and process streaming data in parallel. Streams of events are grouped into logical collections called "topics," which are then partitioned for parallel processing by microservices. Events are processed in the order in which they are received, but remain persistent and available to other consumer services for a specific time period or permanently. Messages may have multiple consumers, and services may perform different functions on the same data. This enables high scalability and flexibility, since services can be provisioned on a mix-and-match basis depending on the needs of the application. Converging file, database, and streaming services via a publish-and-subscribe framework also enables analytical workloads to be constructed consisting of a combination of recent and historical data.

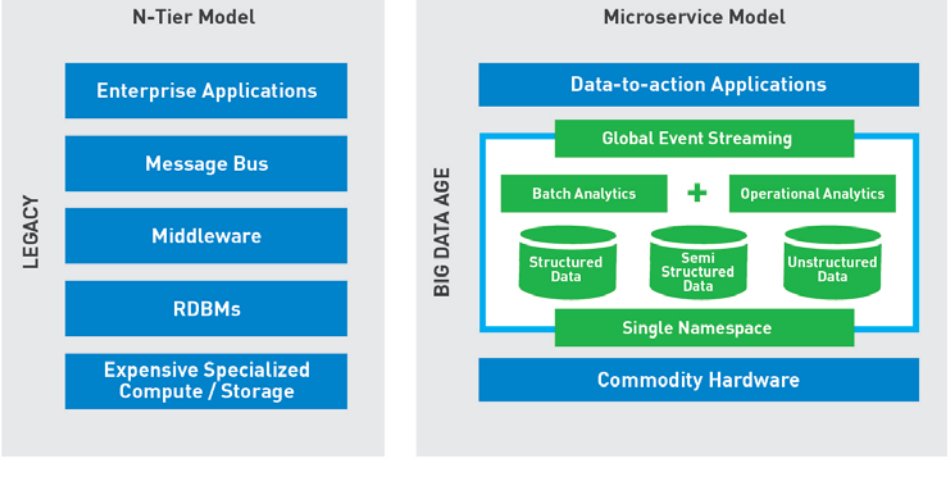
Event sourcing is an architectural pattern in which the state of the application is determined by a sequence of events, each of which is recorded in an append-only event store or stream. For example, each event could be an incremental update to an entry in a database. The entry is the accumulation of events pertaining to that entry. The events can be used to reconstruct a series of transactions by working backwards through the events in the stream. Events can also be ver, microservices can also be called from legacy applications to support new functionality.



Polyglot persistence assumes that applications are evolving to use a variety of data storage techniques, often several within the same application. Rather than force-fitting data to the application, functions are defined as microservices, each of which works on the most appropriate data store. This may include applications which use a combination of structured and unstructured data. A distributed file system manages data access across a wide range of applications defined as microservices while an event store serves as the system of record. All changes to the application state are persisted to the event store which enables the state to be rebuilt by rerunning events in the stream.

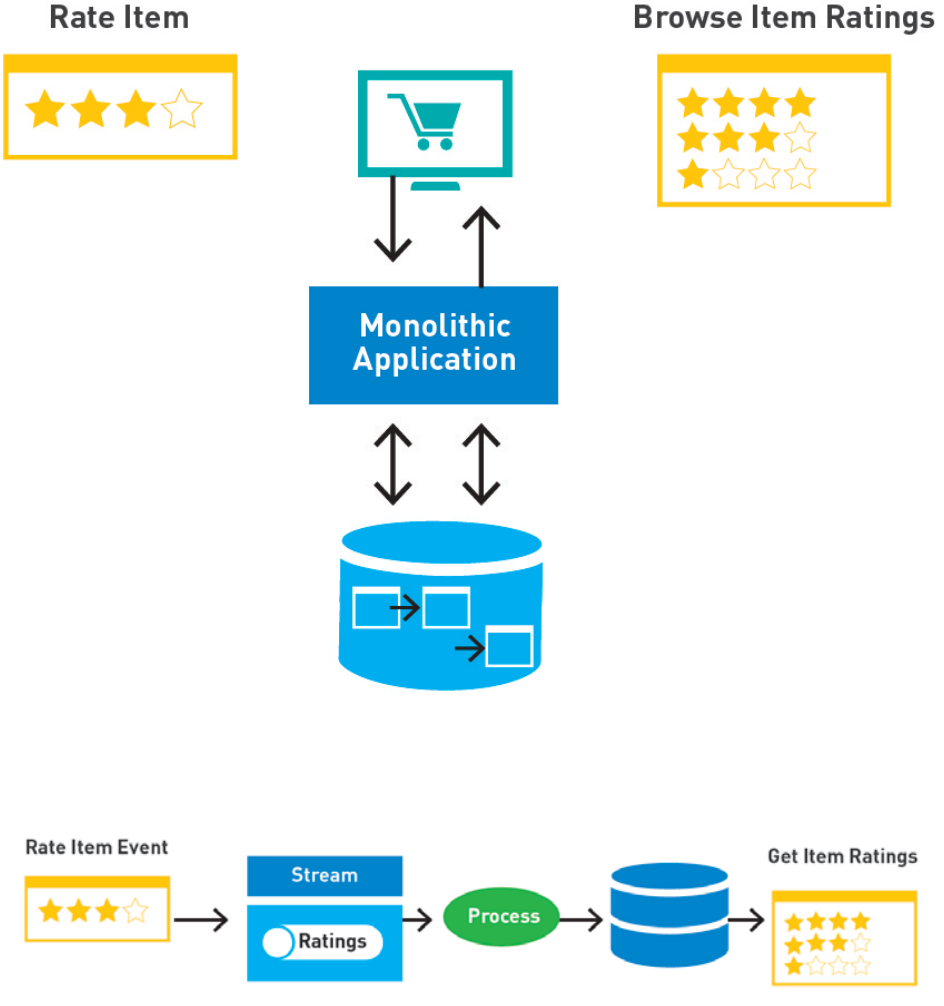


Command and query responsibility segregation (CQRS) separates the read model and queries from the write model and commands, often using event sourcing. Traditional, monolithic applications perform write and read functions on the same database, which creates a bottleneck.



In a CQRS model, reads and writes may be processed separately - even in separate databases – with updates communicated as messages. For example, if a user is looking at a

webpage containing product ratings and submits a review, the change is routed to a separate command model for processing and the result is communicated to the query model to update the webpage. CQRS works well with event-based microservices, but can add significant complexity, particularly when used with monolithic applications.



Microservices and Containers

Containers, which are discussed in the next section, are an ideal deployment mechanism for microservices. Containers are essentially lightweight virtual machines that can be

provisioned with only the infrastructure and management tools that are needed for the service. They can be stored in a library, launched quickly and shut down easily when no longer needed. Because each is self-contained, they can run different services and tools without conflicting with each other.

Containers are ideal platforms for deploying microservices, but they aren't yet ideal for every development scenario. For one thing, they were originally intended to be stateless, meaning that they don't store data locally, but rather retrieve it from somewhere else. Microservices that require persistent, stateful storage require special consideration if they're to be containerized. Examples would be services that process **streaming data or events** in a queue waiting to be written to a database. Complexity increases when different data stores are involved. This can quickly overwhelm the capabilities of a network file share.

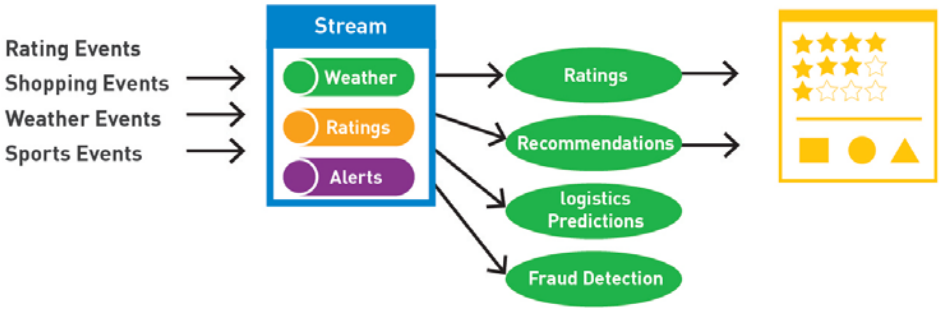
The Innovator's Tips and Tricks, by Quantium

Staffing and skills development are critical to creating a modern data architecture:

- Support and train existing staff on how to use new platforms.
- Hire or contract with people with relevant experience where necessary.
- Necessary skills will likely include Linux, Hadoop, network design and management, systems architecture design, Kubernetes, Docker, Python, Java, Scala and Apache Spark.
- Train people on how to best utilize the new distributed platform. Among the skills you will need are data partitioning, data replication, snapshots, mirroring, Spark query design, YARN, and facility with automation and orchestration tools.

New technologies are rapidly emerging to make containers more appropriate for enterprise-class applications. One of them is Kubernetes, an open-source platform for automating the deployment, scaling, and operations of application containers across clusters of hosts. Originally developed by Google, Kubernetes is a highly functional and stable platform that is rapidly becoming the favored orchestration manager for organizations who are adopting containerized microservices.

New technologies are also coming online to provide flexible, stateful storage. For example, the **MapR Converged Data Platform** can accommodate streams, files and tables into a single file system. It scales smoothly and provides a single platform for functions like authentication, authorization and management. Using a converged platform enables microservices to remain stateless without losing the benefits of data persistence. Such an environment is easier to manage and scale.



Containers

As noted earlier, 40% of respondents to one recent study are already using containers in production and only 13% have no current plans to use containers in the coming year. Datadog **reports** that the average company quintuples its use of containers within the first nine months. For a technology that is barely three years old, that is a stunning adoption rate. Even though issues like persistent storage support and security have yet to be fully resolved, IT organizations are moving ahead enthusiastically.

What are containers, and why the groundswell of support to use them? Containers enable each workload to have exclusive access to resources such as processor, memory, service accounts and libraries, which are essential to the development process. Containers run as a group of namespaced processes within an operating system, which makes them fast to start and maintain. They can be configured to include all of the supporting elements needed for an application, which makes them especially popular with developers. Unlike virtual machines, containers can be spun up in seconds and can be stored in libraries for reuse. They are also portable; an application that executes in a container can theoretically be ported to any operating system that supports that type of container.

Kubernetes has been a big step toward making containers mainstream. Kubernetes introduced a high-level abstraction layer called a “pod” that enables multiple containers to run on a host machine and share resources without the risk of conflict. A pod can be used to define shared services like a directory or storage and expose it to all the containers in the pod. This simplifies the administration of large containerized environments.

Kubernetes also handles load balancing to ensure that each container gets the necessary resources. Kubernetes monitors container health and can automatically roll back changes or shut down containers that don’t respond to pre-defined health checks. It automatically restarts failed containers, reschedules containers when nodes die and can shift containers seamlessly between servers on premises and in the cloud. Altogether, these fea-

tures give IT organizations unprecedented productivity benefits, enabling a single administrator to manage thousands of containers running simultaneously.

Docker is the most popular container platform by a wide margin, but alternatives are available, such as rkt from CoreOS, LXD from Canonical and Azure Container Instances from Microsoft. It's important to note that Container makers have been careful to avoid the standards wars that undermined the Unix market more than 20 years ago. The **Open Container Initiative** is an industry standards initiative that is working to create a basic set of format and runtime specifications that permit interoperability without limiting innovation. It enjoys broad support. A related open-source project called **CRI-O** would put Kubernetes at the center, enabling it to work with any container engine that is compliant with OCI specifications.

In the last year or so, containers have become widely viewed as enablers or optimizers of greater efficiency in DevOps, big data implementations, and microservices. This important new role is attributed to their resource sharing compared to VMs. Instead of waiting hours or longer for virtual machines to be provisioned, developers can outfit their own containers with the infrastructure they require and launch them whenever needed. Containers can also easily be deployed across different platforms, including cloud deployments.

Among their many desired attributes, containers can be launched or abandoned in real-time. This makes them a great match for workloads that are subject to sudden bursts of data activity. A standard VM would be forced instead to undertake a fresh reboot process, which consumes time that containers can spend on actual production work. Containers also raise consolidation benefits to new levels, thanks to the fact that there is no need to essentially reboot the operating system every time a new container is launched. In every virtualized workload, the operating system takes up some portion of the footprint. But there is no requirement for operating system overhead in a container environment. That frees up valuable space for additional memory, processing and other vital development resources.

So not surprisingly, the test and development area is a fertile one in which containers are taking root. As security concerns wane over time, and as container development enables greater capabilities for storing data securely, more and more production workloads will leverage this fast-growing technology. Ultimately, developers using containers will seamlessly move their test and development projects directly into production without major porting efforts. This ability to transfer workloads from one environment to another is destined to become much more important in the emerging hybrid IT environment, in which infrastructure is a combination of existing legacy systems, on-premise and off-premise private cloud, and public cloud.

It's no surprise that containers and microservices have grown in popularity in lockstep with each other; they go together perfectly. Microservices are well-tuned to a container environment because they typically perform a limited set of tasks and are called upon only when needed. Containers are a perfect vessel for microservices. Services can be stored in a library and spun up quickly upon demand, then shut down to be accessed again directly from the library.

As a general rule, containers are stateless, meaning that they don't contain persistent information. When they shut down, any data that was in memory or stored inside the container goes away. Since microservices are miniature processing engines, they typically don't require persistent data. Containers also include all of the support software needed to run the application. This minimizes the risk of conflicts and failure due to other environmental variables. Microservices embedded in containers are self-contained, portable and consistent.

The stateless nature of containers can be a problem in some cases, particularly as the number of instances grows. While it is possible to store data Inside containers, it's not considered a best practice. A better approach is to keep data in a separate data store and then access it upon the launch of the container. Containers enable a wide variety of big data scenarios. For example:

- A web server can run in a container to enable public access to data without risking exposure of sensitive information in adjacent containers.
- That web server can selectively pull user profile data from an RDBMS (SQL) database in another container and combine it with analytics data from a third container running a NoSQL database to deliver individualized shopping recommendations without compromising security.
- Resource efficiency also makes containers good candidates for event-driven applications that use streaming data delivered by **Apache Kafka** or **MapR-ES**. Multiple streams can be processed in parallel and combined for delivery to a **Spark analytics engine**, for example.
- **Machine learning algorithms** running in separate containers can access the same data for different kinds of analysis, greatly improving the speed and quality of results.

Containers are quick to launch, but loading data into containers can be slow by comparison. For that reason, it's tempting to keep a persistent copy of data obtained from, say, a Kafka stream inside the container. The problem is that containers work best when they're stateless, and storing data inside them makes them stateful, or heavy. A profusion of stateful containers can quickly become an administrative nightmare, as well as a security risk.

A better approach is to separate data into a persistent and flexible data store that can be accessed by any container. The problem with that approach is that not all data stores are appropriate to all types of data. NAS filers, for example can't accommodate block storage, and some storage subsystems are too slow to handle streaming data at all.

MapR approaches this problem with its **Converged Data Platform for Docker**, along with a **Persistent Application Client Container (PACC)**. Together, these technologies make it possible for containers to store their operating state upon shutdown and to load any kind of data – including structured, unstructured and streaming data – from a single

persistent store. The approach is linearly scalable and provides a single platform that includes authentication and authorization within one global namespace.

One of the most powerful features of containers is that they can be customized using scripts contained in a Dockerfile, which is a text file that contains all the commands, in order, needed to build a given image. Organizations can start with a basic set of container images that provide basic services for a particular application. For example, a web application container might provide an Apache server, local database and commands for opening a particular server port.

Dockerfile scripts can specify additional resources to be loaded or invoked for a base Docker container depending upon the needs of the application. This reduces complexity by separating the configuration process from the container itself. In general, containers should be kept as simple as possible and modified using Dockerfile scripts at load time. Dockerfile itself provides a limited syntax of just 11 commands that cover most usage scenarios: ADD, CMD, ENTRYPOINT, ENV, EXPOSE, FROM, MAINTAINER, RUN, USER, VOLUME, WORKDIR.

These can be combined to quickly build special-purpose containers, as exemplified in **this ElasticSearch dockerfile**:

```
############################################################
# Dockerfile to build Elasticsearch container images
# Elasticsearch Dockerfile
############################################################

# Pull base image.
FROM dockerfile/java:oracle-java8


# File Author / Maintainer
MAINTAINER Example McAuthor

ENV ES_PKG_NAME elasticsearch-1.5.0

# Install Elasticsearch.
RUN \
  cd / && \
  wget https://download.elasticsearch.org/elasticsearch/elasticsearch/
$ES_PKG_NAME.tar.gz && \
          tar xvzf $ES_PKG_NAME.tar.gz && \
          rm -f $ES_PKG_NAME.tar.gz && \
          mv /$ES_PKG_NAME /elasticsearch

# Define mountable directories.
VOLUME ["/data"]

# Mount elasticsearch.yml config
ADD config/elasticsearch.yml /elasticsearch/config/elasticsearch.yml

# Define working directory.
WORKDIR /data

# Define default command.
CMD ["/elasticsearch/bin/elasticsearch"]

# Expose ports.
#   - 9200: HTTP
#   - 9300: transport
EXPOSE 9200
EXPOSE 9300
```

To build an image based upon this Dockerfile, the user simply types:

```
sudo docker build -t elastic-search
```

To run the instance, the user types:

```
sudo docker run -name es-instance -i -t elastic-search
```

Many organizations are now conducting nearly all of their new development work using containers in order to ensure the maximum portability, scalability and flexibility. Containers are also increasingly being used to host legacy applications. Together with microservices, containers are rapidly becoming an essential building block of agile applications.

The Innovator's Tips and Tricks, by Quantium
An example of tools that enable broader data access:

- Apache Spark as a general purpose distributed data processing and machine learning engine.
- Apache Zeppelin for notebook style development.
- Distributed machine learning engines such as H20.
- Distributed query engines such as Apache Drill.
- Libraries such as XGBoost.

Application Agility

Consider the example of legacy banking applications. Constructed in the age of mainframes for use with green-screen terminals by data processing professionals, these applications must now be accessible to any customer from any device. What's more, they must be intuitive to use. It's no surprise that many banks have spent years overhauling their application portfolios for the age of self-service banking.

Today's business environment permits no such flexibility. Consider one of the new breed of mobile payments apps like Square. Its developers must cope with a constant stream of new devices, payment methods and customer requests. The company closely monitors social media activity to measure customer satisfaction and identify bugs, which it has a few days to fix, at best. The most popular mobile apps are updated weekly in order to maintain feature parity with their competitors. In many cases, those apps must be synchronized with desktop versions that work on any platform and in any browser.

The frenetic pace of business in the age of digital transformation demands maximum application agility. Thanks to cloud computing, the barriers to entry have fallen, and the only way market leaders maintain their positions is by innovating faster than everyone else. Switching costs are low and customers have seemingly endless choices.

Machine Learning

In a recent **National Public Radio interview**, Australian data scientist and entrepreneur Jeremy Howard described how to build a machine learning program that translates back and forth between English and French in near-real-time. “You download a lot of sentences in English or French, write three or four lines of code, let it work overnight, come back in the morning and see if it works,” he said. A programmer can do this without knowing both languages “It’s pretty hard to describe exactly what is done and often I don’t understand at all how my own programs work,” Howard said. He added that he once wrote a deep-learning algorithm that figured out how to spot lung cancer better than a panel of four top radiologists, despite knowing nothing about lung cancer.

Machine learning is the next frontier of application development. The pace of business is speeding up to the point that humans are too slow for some tasks – like live language translation. **Machine learning is a form of predictive analytics** that charges through large volumes of raw data to look for patterns. It continually tests the patterns and tries to assess their relevance to the task, iterating on the good ones and discarding the bad ones. In some respects it’s like the opposite of a query engine. Instead of submitting queries and searching for answers, the machine suggests queries that yield interesting results.

Agile development is a good **use case** for **machine learning**. Libraries of algorithms can be encapsulated in containers and run continuously as processing cycles permit. The results guide developers toward more useful applications. A growing number of machine learning libraries are available as open source. Here are some of the most popular:

TensorFlow - Originally developed by Google for its own internal use, TensorFlow Is highly regarded for its ease-of-use and flexibility. It consists of a Python library that supports a graph of data flows. Nodes in the graph perform mathematical operations, while edges provide data to the nodes. Tensors are multidimensional arrays. Among the notable applications of TensorFlow is Google’s image recognition feature.

Caffe - Caffe is a C++/CUDA deep-learning framework originally developed by the Berkeley Vision and Learning Center. It is especially well tuned for applications related to image recognition and processing. Caffe is optimized for use on graphical processing units (GPU) acting as coprocessors. Machine learning algorithms tested on the GPUs can be easily redeployed into production on the same computer. A new version called CaffeOnSpark enables deep learning models to be deployed onto a single cluster, but be warned that is not production quality.

DeepLearning4J (DL4J) - This is the first commercial-grade, open-source, distributed deep-learning library written for Java and Scala and integrated with Spark. It’s designed for business applications using distributed GPUs and CPUs. DL4J can import neural net models from most major frameworks and provides a cross-team toolkit for data scientists, data engineers, DevOps and DataOps teams.

MXNet - This open-source deep learning framework is primarily used to train and deploy deep neural networks. It’s known for its flexibility and scalability, and it is compatible

with a wide variety of programming languages, including C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl and Wolfram.

Theano - Python developers are the target of this open source project developed at the University of Montreal and named after a Greek mathematician. It's a library for fast numerical computation that can be run on a CPU or GPU, and it's considered a foundational library for creating deep learning models or wrapper libraries. Theano includes a compiler for mathematical expressions and is noted for the efficiency and speed of its code.

Computational Network Toolkit (CNTK) - Microsoft developed CNTK as a way to streamline its own machine learning efforts around speech and image recognition. It's a unified computational network framework that describes neural networks as a series of computational steps in which each node is an input value and each edge is a matrix operation. It can run on anything from a laptop to a cluster with multiple GPU nodes. It was the primary machine learning engine used to build Microsoft Cortana.

Torch - Noted both for its ease-of-use and its flexibility, torch is primarily used for building scientific algorithms quickly and easily. It comes with a large ecosystem of domain-specific packages for such functions as computer vision, signal processing and image recognition, and is considered an excellent choice for use in a GPU-enabled environment using Nvidia's CUDA parallel computing platform and application programming interface model.

PaddlePaddle - Not one to be left out of the party, Chinese search giant Baidu released PaddlePaddle as open-source following similar moves by Microsoft, Google, Facebook and Amazon. Baidu says its toolkit is easier to use than others, but just as powerful and more efficient, requiring only about one quarter the amount of code demanded by other deep learning libraries. Among the applications are machine vision, natural language understanding and route optimization.

Each of these libraries has its own set of tools and processes, which may conflict with each other when running in the same VM. For example, many frameworks are written in Python, which has seen seven new releases since the 3.0 version was introduced in 2008. Trying to run machine learning libraries using incompatible versions of Python in a conventional VM could cause conflicts or prevent programs from running. By containerizing the libraries, each can run in isolation without affecting the others. Containers can likewise moderate differences in CPU requirements, cache, network addresses and other environmental factors that might otherwise threaten stability of the entire system.

That's why NorCom, a full-chain supplier for big data solutions, is using containers as the foundation for the deep learning algorithms it's developing for use in autonomous driving applications. The company uses a purpose-built deep learning framework to efficiently manage massive data sets generated by sensors and cameras in self-driving cars. By running containers on the **MapR Converged Data Platform**, Norcom is getting the speed, scale and reliability to enable multiple deep learning applications to analyze data continuously.

Using Docker containers helps the company scale with agility. The MapR Converged Data Platform's support for multiple deep learning frameworks gave it the flexibility to choose

the best framework for each **use case**. By using a single platform across all data centers, cloud deployments, and edge clusters, the company can quickly roll out containerized **machine learning models** that can be applied to newly created data anywhere in the data fabric. That data can also be immediately added to the training data set because both reside on the same platform.

NorCom is building for the future by planning to support a broad variety of data types, including files, documents, images, database tables and data streams across multiple edge, on-premises and cloud environments. Containers enable IT to support transactional workloads, with high-availability, data protection and disaster recovery capabilities built in[2].

Application Performance Management

IT infrastructure has traditionally been managed from the inside out. The focus was on optimizing the performance of discrete components, such as systems, networks and databases, but without the holistic view of application performance in general. Now, **digital transformation** is driving organizations to expose applications in a multitude of ways, ranging from APIs to web interfaces to mobile apps. Successful big data deployments continue to get bigger and more complex. Microservices, containers and automation tools make developers more productive and applications more agile, but they also introduce new complexity. With new data sources, new **use cases**, new workloads, and new user constituencies, managing growth requires a complete understanding of what is currently happening in the system. This demands a more integrated, top-down approach to management that begins with the user experience and works back to the management of underlying components.

The Innovator's Tips and Tricks, by Quantium

Tips for monitoring a large numbers of applications deployed in containers:

- Log application data for all containers to a path on the distributed file system, then feed those logs to ElasticSearch to enable yourself to build monitoring jobs around log events.
- Use Grafana to create visualizations and to set alerting thresholds.
- Use container health checks in Kubernetes to automatically redeploy containers that become unhealthy.
- Export Kubernetes metrics to monitor for things such as container restart rates and number of applications that are pending launch.

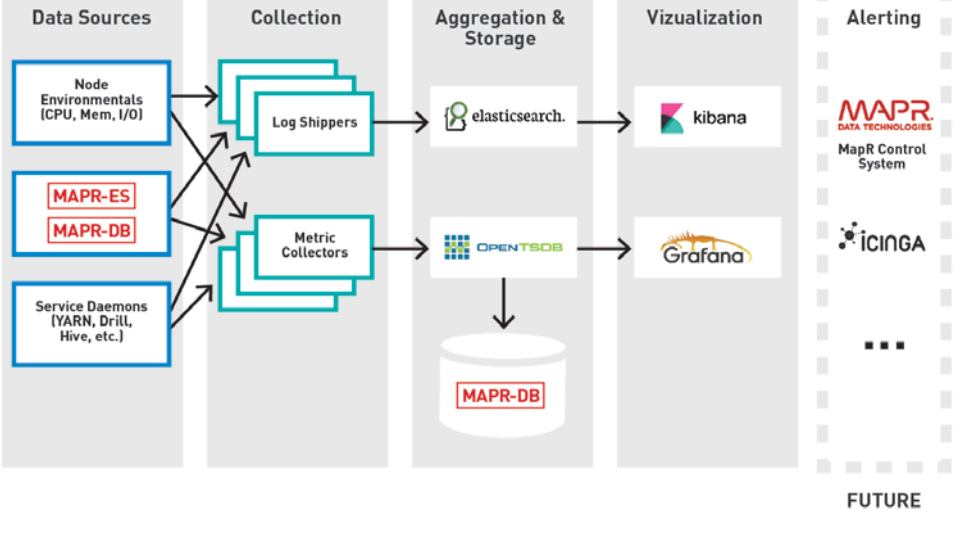
2 **https://mapr.com/company/press-releases/norcom-selects-mapr-deep-learning**

The task of managing applications in a traditional IT environment was simple compared to today's multifaceted and constantly changing workloads. **Hadoop** clusters, for example can become dense as they grow, making it difficult for administrators to pinpoint bottlenecks and outages. An environment today may encompass thousands of volumes and millions of objects. Service levels are impacted by a vastly larger number of people and applications that are accessing corporate networks. The cloud offers relief, but also more complexity.

Administrators must constantly monitor and fine-tune their environments to address questions like these:

- How is my storage usage trending? Do I need to add storage nodes?
- Are all of my cluster nodes evenly utilized?
- Why isn't my service coming up?
- Why is my job slower today than yesterday? Why did it fail?

Application monitoring centralizes the collection of data from across the environment, including logs, metrics, alerts, outages and slowdowns. Administrators can search these logs using tools that excel at leveraging and understanding unstructured data. Typically, monitoring is done from a single dashboard that is customized for the needs of individual administrators. Ideally, the environment also integrates third-party open standards for monitoring.



A robust application performance management system peers into core filesystem and database sources as well as ecosystem components like containers, microservices and or-

chestration tools like YARN and Kubernetes. At its heart is a collection layer that brings together logs from each node and logs them at a pre-configured frequency. Collecting a new metric, or monitoring a new service, is as simple as adding a new plug-in into your metrics collector or log shipper.

Time series databases like OpenTSDB allow millisecond monitoring and historical analysis. Visualization can be provided with open source components like Grafana and Kibana to enable administrators to customize their dashboards.

Say you want to monitor your cluster and you're only interested in looking at CPU and memory across the cluster. You can add that to a dashboard, and that should be it. But if you want to look at all of the information on a particular node, you can tag that particular node into your system and look at memory, disk, and CPU next to each other for that particular node. Not only can you build customizable dashboards, but if you have your own environment for monitoring other components of your infrastructure, you can easily integrate it using the APIs that OpenTSDB and Elasticsearch provide. That way, you have a single pane of glass to look at everything within your monitoring environment.

Microservices Performance Management

Microservices introduce an additional wrinkle to application performance management because each service has its own set of resources, dependencies and overhead requirements. A microservices-heavy environment may require many remote calls, each dealing with issues such as network latency, queues, connection reliability and workloads. There are also overlapping interdependencies. For example, if microservice X relies on output delivered from microservice Y, a slowdown in Y may appear as a problem in both microservices, even though only one is to blame.

The best way to troubleshoot such complex environments is with logs, stream processing engines and search engines like Elasticsearch that handle unstructured data well. It's important administrators understand the interdependencies of all microservices in production in order to determine the root of a performance problem. One way to help downstream efforts is by leveraging build and deployment automation tools such as Jenkins, Chef and Puppet. Dependencies can thus be mapped prior to deployment, making troubleshooting easier. Automation can also be used to constantly test applications in the same way a user would to look for early signs of developing problems. These workloads can then be failed over automatically or an alert sent to an administrator for intervention. Also keep in mind that every opportunity to use an event stream is a way to identify traffic flow which later enables replayability when performing root cause analysis.

Container Performance Management

The process of **managing containers** is quite similar to that of managing microservices, but the variables are different. For example, an administrator needs to know how many containers are running, the CPU and memory utilization of each and the health of the network they're running on. Performance management requires knowing the communication

dependencies between individual containers, the image that's deployed in individual containers and the services that run there.

Because containers can be spun up and shut down quickly, traditional performance management technology is less effective in these environments. Stateless, containerized applications benefit from a persistent and comprehensive data services layer to not only provide resilient storage for containers, but also the database and messaging/streaming capabilities that many containerized operational applications require. Fortunately, Docker provides rich APIs for such functions as starting and running containers, creating logs and metrics and managing images, networks and volumes. However, the ecosystem of management tools is still incubating, forcing many companies to roll all or part of their own solutions.

In sum, achieving the goal of application agility demands an environment that is unconstrained by the traditional limitations of systems management. Developers need the flexibility to launch instances when they need them, not when IT says they can have them. Those instances should contain the tools developers need to do their work, not just the tools that the existing infrastructure can support. Tasks that used to require manual intervention now can – and should – be automated. Seamless orchestration across multiple servers should be assumed. All of these capabilities are available today for organizations that are ready to embrace them.

The Innovator's Tips and Tricks, by Quantium

When moving to containers, use a repository like Artifactory, or Nexus. This enables you to maintain a repository of containers in git, and upon committing to master you can build a new image via Jenkins and then push it to the repository.

For container deployment, take an app store approach. Users can self-provision applications onto the platform from the app store, as well as destroy and restart applications.

5 Infrastructure Agility

Application agility depends upon an equally agile infrastructure. Both have similar characteristics. As we've seen, agile applications are deconstructed, distributed and dynamically assembled as needed. The whole point of microservices and containers is to eliminate as much of the need for vertically integrated and hard-coded logic as possible. It is also to remove dependencies on specific servers and hardware components.

Today, organizations have an unprecedented variety of options for deploying infrastructure. There's traditional on-premise infrastructure – the data center – public cloud, private cloud and hybrid cloud. There is also a huge selection of software-as-a-service (SaaS) providers who expose their services through APIs that their customers can use to extend their own applications. The downside of choice, of course, is complexity. That's why agile infrastructure should be designed to be deployed on as broad a combination of platforms as possible. Resources should be shared. All platforms should support multi-tenancy for the greatest deployment flexibility. This is even desirable within the on-premises data center. As much as possible, platforms should have common operating systems, containers, automation tools, permissions, security and even pathnames. If something is in a home directory on-premise, there should be a duplicate home directory and path on a cloud platform so applications don't come to a halt over things like simple naming conventions.

Deployment Options

Deployment options fall into four basic models, which can be mixed and matched as needed.

On-Premises Infrastructure

This is been the dominant enterprise computing model for more than 50 years. Organizations maintain their own equipment and software in a captive data center with full control over all aspects of processing, scheduling, administration and maintenance. Many organizations in regulated industries have no choice but to use an on-premises model because of the need to tightly control data and document processes. However, the cost and significant capital expense involved with building and maintaining on-premises architec-

ture is prompting many organizations to shift some or all of their workloads to more-flexible cloud options. On-premises computing won't go away anytime soon, however. Legacy equipment and applications may be incompatible with a cloud environment, and organizations that want to protect investments in hardware and software may choose to maintain on premises investments for years until depreciation cycles have run their course and applications can be redeveloped.

Public Cloud

Public cloud makes resources, such as processors, memory, operating systems, applications and storage, available over the public internet on a pay-per-usage basis. Think of it as a computers in the sky. Public cloud is like using a local server, but the server is virtualized and managed elsewhere by a cloud provider with a high degree of automation.

Organizations use public cloud for a variety of reasons, but the most popular are flexibility, scalability and ease of administration. Public cloud instances can be launched with a few mouse clicks and just as easily taken down when no longer needed. Developers and end-users can, in many cases, deploy their own cloud instances without approval from IT and its accompanying delays. Billing is usually based upon usage, which gives organizations accountability and flexibility to pay only for the resources they use. Public cloud instances can be scaled up or down with relative ease, and many cloud providers offer best-of-breed automation tools to make administration easy. Public cloud is also an excellent platform for developing applications that will "live" in the cloud, such as those meant for use on mobile devices or with services that are exposed via APIs.

Private Cloud

For organizations that want the flexible automation benefits of public cloud but need to keep resources on premises for control or compliance reasons, private cloud is a popular alternative. This model provides the same scalability, automation and flexibility advantages of public cloud and on-premise environment that can be physically secured and tightly managed. Private clouds can be built using existing data center equipment architecture or licensed from public cloud providers, could deliver what is essentially a version of their existing services in a secure environment. True private cloud is more than just virtualization. The research firm Wikibon **defines** it as encompassing converged architecture, virtualized software and hardware, self-service provisioning, orchestration/automation and a single point of control.

Hybrid Cloud

When you combine a public and private cloud, you get a hybrid cloud. This architecture combines both models in a manner that is seamless and that permits workloads to easily move back and forth. This gives organizations a combination of control and flexibility that can be adjusted to the situation. Hybrid architecture preserves existing hardware and soft-

ware investments while giving companies the flexibility to move applications to the cloud as resources and budgets permit. Not all applications can be moved easily, and some may continue to live for a long time in private data centers. In those cases, organizations may opt for a “cloud bursting” approach in which demand spills over to a duplicate or compatible cloud application as needed. This reduces the need to add on-premise infrastructure that sits idle much of the time. There are even cloud-cloud options, in which applications move back and forth between multiple public clouds.

Containers and Clouds

One of the most compelling advantages of cloud computing is developer productivity. As noted above, developers can quickly spin up their own cloud instances, provision the tools they want and scale up and down easily.

Containers are an ideal tool for developers to use when shifting between on premises, private cloud and public cloud architectures. Because containers are independent of the underlying operating system and infrastructure, they can be moved quickly and with minimal disruption. Some organizations even use multiple public clouds, and shift workloads back and forth depending upon price and special offers from the service providers. Containers make this process simple.

Orchestration

Agile infrastructure should minimize the need for human intervention in routine tasks, such as resource deployment and management. Overworked IT administrators and paperwork can introduce significant delay that undermines the value of cloud environments. Automation makes cloud computing fast and efficient by using software tools to handle these tasks.

For example, automation can enable the setup of multiple virtual machines with identical configurations using a single script written in Puppet, an open-source configuration tool that enables applications and infrastructure to be defined using English-like commands. Puppet scripts can be shared and used to enforce changes across data center and cloud platforms.

Ansible is an open source automation platform that can be used for tasks like configuration management, application deployment and task automation. It can also be used to automate cloud provisioning and intra-service orchestration using a “playbook” metaphor that permits multiple automation tasks to keep the combine to powerful effect.

As noted earlier, Kubernetes is bringing these same kinds of automation and orchestration capabilities to containers, with features that are customized for the unique stateless, self-contained characteristics of those vehicles. Kubernetes is optimized for orchestrating large numbers of containers, ensuring that each has the resources it needs and providing for things like health monitoring, restart and load balancing.

Kubernetes isn't a replacement for Puppet and Ansible, but is another resource that works specifically at the container layer and that can be managed by those automation tools. The combination of VM automation and Kubernetes gives IT organizations unprecedented productivity advantages compared to manual systems administration.

The Innovator's Tips and Tricks, by Quantium
Use Kubernetes as the orchestrator of your container/microservices architecture. Although newer than some alternatives, it has better functionality and larger community supporting and enhancing it.

Edge Computing

The internet of things will create vast new volumes of data, a flood that International Data Corp. **expects** will reach 44 zettabytes annually by 2020. To help visualize that, if you covered a football field with 32 gigabyte iPhones and kept stacking layers on top of each other, by the time you got to 44 zettabytes the stack would reach 14.4 miles into the air. At that altitude, the temperature is -65° and the barometric pressure is 1/30 that of the surface of the earth. IDC further estimates that machine-generated data will account for 40 percent of the digital universe in 2020, up from 11 percent a decade ago.

These unprecedented data volumes will require a new approach to processing, since traditional server, storage and network models won't scale enough. This is why edge computing is rapidly emerging as a new architecture.

Edge computing distributes resources to the far reaches of the network and close to the devices that generate data. Edge servers collect streaming data, analyze it and make decisions as necessary. These servers can pass selected or summary data to the cloud over the network, but most of the processing takes place locally.

Edge computing has some important implications for IT infrastructure and application development. Many applications will need to be restructured to distribute logic across the network. Storage will likewise need to be decentralized. This will create new issues of reliability and data integrity that are inherent in broadly decentralized networks. Cloud servers will become control nodes for intelligent edge devices, performing summary analytics while leaving real-time decision making to edge servers.

Containerized microservices will be an important technology in the construction of IOT backplanes. Distributed processing frameworks will require federated, multi-domain management with intelligence moving fluidly to the places it's most needed. Automation and orchestration tools like Kubernetes will evolve to meet this demand.

Serverless Computing

Cloud computing has made servers transparent, and serverless computing – also called event-driven computing, or Function-as-a-Service (FaaS) – takes this to another level. It reimagines application design and deployment with computing resources provided only as needed from the cloud. Instead of being deployed to a discrete server, containerized, microservices-based routines are launched in the cloud and call upon server resources only as needed.

The idea is to remove infrastructure concerns from the code, thereby enabling microservices to interact more freely with each other and to scale as needed. The user pays only for server resources as they are provisioned, without any costs associated with idle capacity.

Amazon Web Services' **Lambda** is an example of serverless computing. It's used to extend other AWS services with custom logic that runs in response to events, such as API calls, storage updates and database updates.

While still a fledgling technology, serverless computing has great potential to enable the development of applications that are far more scalable and flexible than those that are bound by servers or VMs. Containers and microservices will be key to the development of this new model.

Security

Having a robust but flexible security architecture is integral to the success of these technologies. The use of containers and microservices may significantly increase the number of instances running in your organization compared to virtual machines. This requires attention to security policies and the physical location of containers. Without proper security, you would want to avoid for example, running a public web server and an internal financial application in containers on the same server. Someone who compromises the web server to gain administrative privileges might be able to access data in the financial application.

Containers increase the complexity of the computing infrastructure because they can be dynamically orchestrated across services or even across multiple clouds. Self-provisioning means that administrators don't necessarily know which containers are running, and a container's IP address may be invisible outside of the local host.

Containers are different from virtual machines in the area of security. They use similar security features to LXC containers, which are an operating-system-level virtualization method for running multiple isolated Linux systems on a control host using a single Linux kernel. When a container is started, it creates a set of namespaces and control groups. Namespaces ensure that processes running within a container cannot see or interfere with processes running in other containers. Each container also has its own network stack, which prevents privileged access to the sockets or interfaces of another container.

Containers can interact with each other through specified ports for actions like pinging, sending and receiving packets and establishing TCP connections. All of this can be regula-

ted by security policies. In effect, containers are just like physical machines connected through a common ethernet switch.

Stateful containers present somewhat more complicated security considerations because they connect directly to underlying storage. This presents the possibility that a rogue container could intercept read or write operations from a neighbor and compromise privileges.

Using a persistent data store with security built-in can minimize risk. The data store should have the following features:

- **A pre-built, certified container image with pre-defined permissions.** This image should be used as a template for any new containers so that new security issues aren't introduced.
- **Security tickets.** Users can pass a MapR ticket file into the container at runtime with all data access authorized and audited according to the authenticated identity of the ticket file. This ensures that operations are performed as the authenticated user. A different ticket should be created for each container that is launched.
- **Secure authentication at the container level.** This ensures that containerized applications only have access to data for which they are authorized.
- **Encryption.** Any storage-or network-related communications should be encrypted.
- **Configuration via Dockerfile scripts.** This can be used as a basis for defining security privileges with the flexibility to customize the image for specific application needs.

Microservices bring their own brand of security challenges. Instead of protecting a few monolithic applications, administrators must attend to a much larger number of federated services, each communicating with each other and creating a large amount of network traffic.Service discovery is a capability that enables administrators to automatically identify new services by pinpointing real-time service interactions and performance.

Cluster-based micro-segmentation is another useful tool. Network segments can be set up with their own security policies at a high level – for example, separating the production environment from the development environment – or in a more granular fashion, such as governing interactions between a CRM system and customer financial information. These policies are enforced at the cluster level.

Automation is also the security administrator's friend. The complexity of a containerized microservices environment naturally lends itself to human error. By using automation for tasks such as defining policies and managing SSL certificates, that risk is significantly reduced.

In the early days of the container wave, security was considered to be a weak point of the technology. Much progress has been made in just the past two years, though. By using the techniques noted above, your containerized microservices environment should be no less secure than your existing VMs.

The Innovator's Tips and Tricks, by Quantium
Users who can launch containers effectively have root access. Use container templates that provide a strict level of security. Build a framework that limits the actions a user can perform on applications. For example, you can control which containers users can deploy onto the platform through the use of an app store. Users must be given explicit permission to deploy containerized applications onto the platform.

Buyer's Guide to a Modern Data Architecture 6

As you begin the process of moving to a modern and **agile data architecture**, keep a set of operating principles and objectives in mind to minimize complexity and future-proof your choices.

Walking the Walk: Quantium's Journey to Developer Agility

Data analytics developer **Quantium** has embraced many of the principles and technologies described in this book, and its development and IT teams are seeing the payoff in improved business agility, better decision-making and faster results.

A few years ago, the maker of advanced analytics made the decision to move much of its development work from a Microsoft SQL Server environment to a **MapR** big data platform using containers. The idea was to give developers more control over their own environments while enabling innovations to be easily shared.

"The old way of coding would be going to the IT department and asking them to spin up a VM. You'd have to wait a week for that," said Gerard Paulke, a platform architect at Quantium. "If you used up all the RAM, you'd have to ask for another VM. Managing resources was very difficult."

Shared infrastructure had other shortcomings as well. If a VM went down, so did all the processes running on it, and there was no guarantee that other VMs would be able to pick up the load. Version control was a chore. Developers couldn't use the latest versions of their favorite tools until they had been installed and tested by the IT organization. And upgrades could break software that had been created to work with earlier versions of those tools.

Containers now provide much of the functionality that was formerly served by virtual machines. Developers have the freedom to not only launch their own environments whenever they want but also to work with their preferred tools without interfering with others. "For example, we can have multiple versions of the Hive metastore in different containers," without causing conflicts, Pauke said. "It's agile and resilient."

Quantium created a common base Docker image that has all the components needed for secure development. Developers can use these shells as a template for constructing their own environments.

"If developers want to try something new, they can just spin up an edge node," Paulke said. "If they like it, we can containerize it and put it into our app store for anyone to launch." Sharing containers enables everyone to benefit from each other's innovations.

Automation and orchestration tools have taken over from human administrators to handle the deployment, scaling, and management of containerized applications. Apache Mesos and Apache Marathon, which are precursors to Kubernetes, provide isolation and resource allocation so containers don't conflict with each other. If a VM fails, any running containers are automatically shifted to another VM. Orchestration software also automatically finds the resources any given container needs so that it launches smoothly.

"From the user's perspective, their services are always available," Paulke said. "We're running hundreds of applications, and literally one person can manage the whole infrastructure."

For others who are interested in adopting containers, Quantium advises designing the platform to be self-service from the ground up. Use a basic set of common container images that developers can check out of a library, and expose as much as possible through well-documented APIs.

Bare-metal servers should be identically configured to the greatest degree possible so that automation can be handled smoothly in software, using tools like Puppet and Ansible. Use playbooks to track changes to the environment and enable backtracking to earlier versions, if necessary.

Finally, talk a lot and listen even more. In moving to an agile environment, "I've found people issues are the biggest issues," Paulke said. Developers need to get comfortable with the idea of self-service, and IT administrators must learn to give up some control. Once they see how fast and flexible the new world of development can be, however, they won't want to go back.

Implementation Considerations

Backups, disaster recovery and business continuity. In the age of new technologies this tends to be the most overlooked feature because it is generally ignored until too late of a stage in an implementation. Ensuring that the core platform handling the storage of the data can be properly backed up and restored are critical to the longevity of any business. Backing up massive volumes of data is often easier said than done. When raising the question of backing up a Hadoop cluster at a large international conference, only one person out of 100 said they had tested backing up and recovering their platform. Make sure that your research includes this topic as the number one most important capability.

Minimize data movement. As the internet of things brings more data generating devices online and overall data volumes continue to swell, you'll want to think about the most appropriate places to process data in order to minimize bandwidth usage and its accompa-

nying network latency. Intelligent edge devices will increasingly be critical to efficient design. Avoid centralizing all your processing in a few servers. When working with cloud vendors, use service-level agreements that specify response time thresholds. Be aware of surcharges for data transfer between on premises and cloud environments.

Unified/flexible security model. As noted earlier, containers and microservices introduce complexity, which can lead to security vulnerabilities. Adopt a policy-driven governance structure, use group-based authentication and minimize colocation of production, development and public facing services. Automate as much as you can through scripting and orchestration services like Kubernetes.

Establish data governance practices. Data governance procedures ensure that data is available, usable, valid and secure. A data governance program typically includes a governing body, a defined set of procedures, ownership criteria and execution plans. A policy specifies who is accountable for data, and processes are established that define how the data is stored, archived, and protected. The policy also specifies how data can be used and by whom. If government regulations are involved, that is included as well. Having a mature data governance practice in place enhances security, reduces duplication and creates a foundation for use and growth of data.

Establish data lineage and auditing procedures. As data flows through a process from capture to storage to refinement to use in production or analytic applications, it's important to track and audit its status at all stages in order to prevent errors or omissions. Data lineage provides an audit trail of data points at each stage, with the ability step back through the process for debugging or error correction.

Apply Master Data Management. MDM is a technique for linking all critical data in an organization to one master file that serves as a common point of reference. With MDM, you have one canonical reference to critical information that points to all related data, regardless of where it resides. Access to that information can then be granted on a need-to-know basis without regard to organizational boundaries. MDM can be applied across cloud and on-premises data. The discipline minimizes errors and duplication, provides employees with the most up-to-date data, simplifies ETL and reduces compliance risk.

Support multiple data formats. Data can no longer be assumed to fit neatly into rows and columns, or to conform to standard formats. As new data sources like the internet-of-things come online, data formats will proliferate. Consider your future needs to process such data types as images, audio/video, geolocation, temperature and geometric. Data stores should accommodate the JSON data interchange format for flexibility and readability.

Build scalable infrastructure. A scale-out architecture gives you maximum flexibility to expand your resources as needed. It should be your architectural choice for servers, storage and networking equipment. Scale-up architectures are sometimes needed for performance-intensive uses, such as network switching, but they impose limitations that can be expensive to resolve if you run out of capacity.

Allow for geo-distribution. Many organizations are decentralizing data analysis, enabling regional teams to maintain independent databases that are periodically consolidated

with the head office. Low-latency geo-distribution of very large-scale databases and computation engines can provide a competitive edge, but only if good data governance principles are in place. The architecture must provide persistence across multiple data types, with data shared and updated everywhere at the same time with fine-grained access control. Containers and microservices are well-suited for a geo-distributed approach.

Use Standard APIs. Avoid hard-wired interfaces where possible and instead use APIs. This makes it easier for your organization to share application functionality and for you to selectively expose functions to others. Where possible, use standardized APIs such as those specified by **OpenStack**, **Windsoc** or the frameworks of your preferred programming languages.

Support containerization. We've already made the case at length for adopting containers to take advantage of their flexibility, speed of deployment, configurability and automation. Containers should be a standard part of any agile infrastructure.

Support SQL across all data formats. SQL is the lingua franca of data access, and it is supported by nearly every major database platform, either natively or by extension. Not every product that says it is SQL-compatible conforms to a full set of SQL commands, however, and there are also multiple versions of SQL to consider. Define a base set of query functions you will need and ensure that they are supported by any database management system you bring into your environment. ANSI SQL is the key to compatibility.

Support multiple machine learning frameworks. Machine learning is in its early stages of maturity, and there are numerous libraries available, as was described in Chapter 4. Each of these libraries has its own strengths and weaknesses, so ensure that your environment can accommodate a variety of libraries as your needs evolve. File system standards are the key.

Support on-premises and cloud infrastructure. Give yourself as much room as possible to decide which platforms to use for which applications. If adopting a private or hybrid cloud infrastructure, choose platforms that are compatible with those of public cloud providers so that you may easily shift workloads. This approach also enables you to move on-premises applications to the public cloud in stages by first testing them on a private cloud.

Budget for training. Many of the technologies described earlier in this book will require significant **retraining of existing staff**. This can be costly, but it is usually less expensive to retrain than to hire new developers, who don't have full knowledge of your business. Perform a knowledge assessment as part of your transition planning and develop a timeline and budget for upgrading necessary skills.

Build a DevOps and DataOps culture. When moving to a DevOps and DataOps development approach, be prepared for significant disruption as the organization adopts agile processes. This is a major departure from traditional waterfall lifecycles, and education will be required of both developers and users. Developers must acclimate themselves to more rapid code releases and greater control over their environment. Users must be ready to engage more closely with the development process as code reviews become more frequent. This is a cultural shift that often requires the IT organization to play the role of the evangelist in selling the benefits of DevOps and DataOps across the organization.

Prepare an application migration plan. Moving existing applications to cloud infrastructure can involve as much work as rebuilding them from scratch. Analyze your application portfolio and determine the degree of difficulty involved in re-platforming. Some legacy applications may be best left to on-premises infrastructure until replaced. Some may be better candidates for rebuilding as microservices in the short term. Others may move smoothly to the cloud with little modification. If your plans call for rebuilding any existing applications, always consider a microservices approach.

Define administrative tasks. Automation and DevOps can have significant impact on tasks like systems administration, backup and security. Many tasks that were once performed manually can be automated. Backup may become more complex in a heavily containerized environment. As noted earlier, containers and microservices also introduced new security concerns. In most cases, moving to agile infrastructure reduces overall systems management overhead, but it doesn't reduce the need for new skills. Plan your training calendar and budget accordingly.

Assess network impact. Containers and microservices don't necessarily increase network overhead, but network performance can be affected if load-balancing and resource optimization aren't effectively applied. Move to containerization in stages and use your test environment to measure the impact on networks. Your network managers may require some retraining.

Adopt orchestration. As noted earlier, anything greater than a modest container deployment benefits from orchestration. Tools like Kubernetes will leave behind the use of manual provisioning and management and give you the flexibility you will need to scale your containerized environment in a highly automated fashion.

Developing the Business Case for a Modern Data Architecture

As many organizations moved to a modern data architecture, they are using what can be called a "connected" approach. For example, they may use **Hadoop and Spark for analytics**, **Kafka** for **streaming**, **HBase** or Cassandra for operational data stores and MongoDB for document-oriented processing. Data sources may include MQTT from lightweight IoT devices, JSON, structured log data, and traditional relational tables. Each engine is assigned a subset of available nodes and hard-coded interfaces are used to transfer data between services. This approach is common in environments that have added new data types in a piecemeal fashion, or where ownership of data resides in different departments.

An alternative architecture is what can be called a "converged" approach. In this scenario, all services can run anywhere in a cluster and share resources. There may be different topics in a stream that need to read or write data from different sources, but these topics all reside in the same place. The advantage of this approach is that any application can consume data from any source and all resources are shared because they live on the same cluster.

The converged approach has several major advantages over the connected approach, including that of linear scalability of the resource pool. Shared infrastructure reduces overhead and simplifies management. A unified data platform enables greater application flexibility and streamlines the use of microservices. Data can be persisted as part of the platform through APIs. For example, JSON documents can be persisted directly into the database without connecting code because they're on the same platform.

A converged platform is superior to a connected one in nearly every case, but moving from a connected to a converged platform is not always simple for some organizations. The lines in a connected architecture are a substantial work effort that need not be present in a converged architecture.. Many of the advantages of containers, microservices and orchestration may be lost when interfaces need to be hand-coded and manually maintained. Developing the business case for a **modern data architecture** starts with making a commitment to adopt converged principles.

The Innovator's Tips and Tricks, by Quantium

Service-oriented architectures have been attempted for years and always failed to reach their potential. What's different this time?

- Microservices are self-contained and easy to deploy, scale and move.
- Containers enable you to isolate resources, such as CPU and storage, and therefore achieve better resource utilization over statically partitioned services.
- Containers also isolate software dependencies, permitting more flexibility for developers and operations.

Making a business case starts with understanding the stakeholders and their priorities. If the initiative is driven entirely by the IT organization, it will probably fail. Understand the business context. Is the company in growth mode or is it focused on keeping costs down? Is IT seen as a source of competitive advantage or as a cost of doing business? Is the company attempting a digital transformation or is it content to stay the course? Is management's vision long-term-oriented or quarter-to-quarter? Organizations that are in cost-control mode or unable or unwilling to embrace change will struggle to make the commitments needed to transform the application architecture.

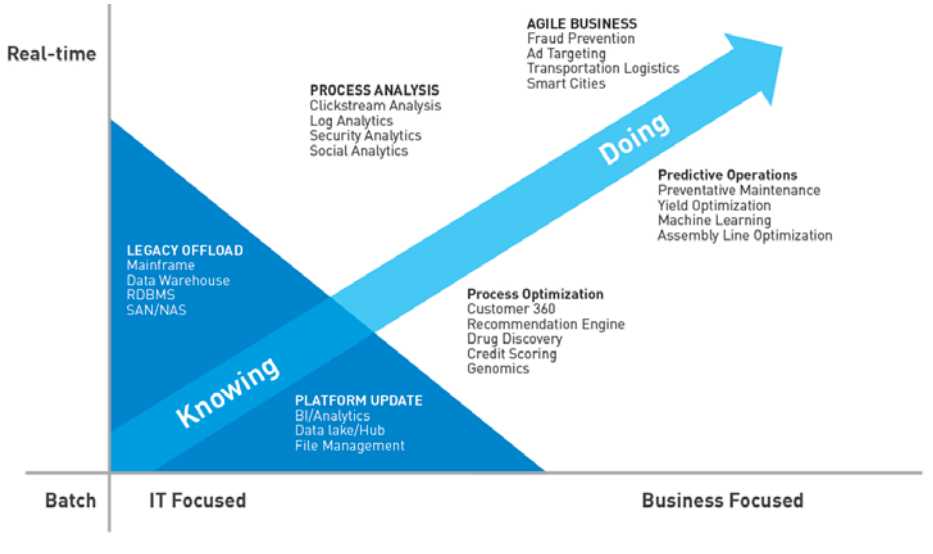
One company that built a strong business case was Karlsruhe Institute of Technology (KIT), one of the largest and most prestigious research and education institutions in Germany. KIT's challenge was to develop new technologies to run control centers for renewable energy networks. Because many more devices are active in the creation of renewable energy, the application needed to be able to scale and seamlessly integrate new data for-

mats, particularly those generated by the internet of things (IoT) devices. KIT sought to create an ecosystem of big data storage applications using **Apache Kafka**, **Spark** and **Apache Drill** for processing high-velocity data, combined with microservices-based applications that would be integrated into future control centers as its infrastructure expanded.

By adopting the **MapR Converged Data Platform**, the organization has achieved unprecedented flexibility. The IT organization can execute any file-based application directly on data in the cluster without modification. Both legacy applications and new microservices-based applications can access the same data.

KIT was also able to break down some legacy applications and split them into smaller, scalable parts using microservices and containers. The MapR platform enabled it to combine container automation and big data software on one computing cluster for better performance and data integration.

Ultimately, KIT will build an intelligent national IoT infrastructure that can also be used in other projects that combine high data velocity and volume. By selecting a high-performance platform that supports dedicated services on top to store and analyze data, the Institute is paving the way for a future free of reliance on fossil fuels[1].



2

There are four stages to building a business case for a modern data architecture.

Stage 1: Define the business issue. As noted above, companies in cost-containment mode have very different motivations for considering a modern data architecture than

1 **https://mapr.com/resources/karlsruhe-institute-technology-uses-mapr-build-control-centers-renewable-energy**

2 **https://mapr.com/ebooks/architects-guide**

those that are undergoing a digital transformation. **Making a business case** involves understanding where the greatest perceived value will be. A modern data architecture has significant potential both to cut costs and to transform the business, but implementation approaches are quite different.

For example, a company seeking to grow its business can use modernization to unshackle itself from the limits of legacy applications and database platforms, develop applications faster and more nimbly and transform the customer experience. It will probably want to undertake a more sweeping overhaul of its IT infrastructure than a company that is looking to reduce licensing costs by moving to open software, for example. In the latter case, a more modular approach should be pursued, looking for quick wins and incremental growth.

Stage 2: Analyze alternatives and select the best options. Many factors go into selection criteria, including timeframe, budget, technical abilities of current staff, robustness of existing infrastructure and the need to reengineer legacy applications. Rolling your own solution using open source tools is generally the lowest-cost option, but it can place heavy demands upon existing staff. Buying packaged solutions from vendors is more costly but also quicker and less risky. Or you can attempt some combination of the two. Many organizations choose a core set of open source tools – such as **Hadoop**, **Apache Spark**, **Apache Drill** and Docker – and then select an integrator or software vendor that can deliver a platform to support them. Whatever your approach, be sure you are thinking of the long-term and planning for scalability and adaptability to new data sources and applications such as deep learning.

Stage 3: Prepare the business case. Modernizing your data architecture can require a lot of persuasion. Significant cost and time may be involved before benefits are evident. Rather than risk the project on a single presentation to the executive committee, meet individually with key stakeholders in advance to test your case. Ask for feedback on how to best present your case and anticipate the questions and objections you will encounter.

When preparing the case, keep the business objectives in mind. Don't fall into the trap of making this an IT-centric proposition. Align outcomes with the business goals, providing as much detail as possible about actual anticipated costs, revenues and overall ROI. Lean on business stakeholders to help with this process. Be prepared to present three different cost scenarios: best case, most likely and worst case. Also be prepared to defend your assumptions. An executive committee or board of directors will want your facts and figures to be grounded in reality. Integrators or vendors can help, since they have engaged in similar projects many times.

Stage 4: Deliver the business case. If you have prepared as described above, delivering the case is the easy part. Don't let your enthusiasm get in the way of making your argument. Your audience will most likely be skeptical and will attempt to poke holes in your model. Thoroughly researching the costs, timelines and paybacks is your best defense.

Final Thoughts

Building applications based upon services isn't a new idea, but until now structural impediments have prevented organizations from realizing the benefits of this architecture. The combination of software-defined infrastructure and cloud computing has now removed the major obstacles, setting the stage for the most significant transformation in the way applications are built and delivered in the past 50 years.

The monolithic, vertically integrated applications that have defined enterprise computing since the 1960s will increasingly become a liability for companies seeking to realize the agility that will be demanded by digital transformation. All organizations that hope to operate at the speed of business in the future will need to adopt a services model. The only question is when.

The technology tools are now in place. The benefits of a DevOps and DataOps approach are clear. While it may not make sense for organizations to be diving in headfirst at this stage, there is no reason not to begin experimentation and long-term planning. The risks of not doing so are too great.

About the Author

James A. Scott (prefers to go by Jim) is Director, Enterprise Strategy & Architecture at MapR Technologies and is very active in the Hadoop community. Jim helped build the Hadoop community in Chicago as cofounder of the Chicago Hadoop Users Group. He has implemented Hadoop at three different companies, supporting a variety of enterprise use cases from managing Points of Interest for mapping applications, to Online Transactional Processing in advertising, as well as full data center monitoring and general data processing. Jim also was the SVP of Information Technology and Operations at SPINS, the leading provider of retail consumer insights, analytics reporting and consulting services for the Natural and Organic Products industry. Additionally, Jim served as Lead Engineer/Architect for Conversant (formerly Dotomi), one of the world's largest and most diversified digital marketing companies, and also held software architect positions at several companies including Aircell, NAVTEQ, and Dow Chemical. Jim speaks at many industry events around the world on big data technologies and enterprise architecture. When he's not solving business problems with technology, Jim enjoys cooking, watching-and-quoting movies and spending time with his wife and kids. Jim is on Twitter as ***@kingmesal****.*