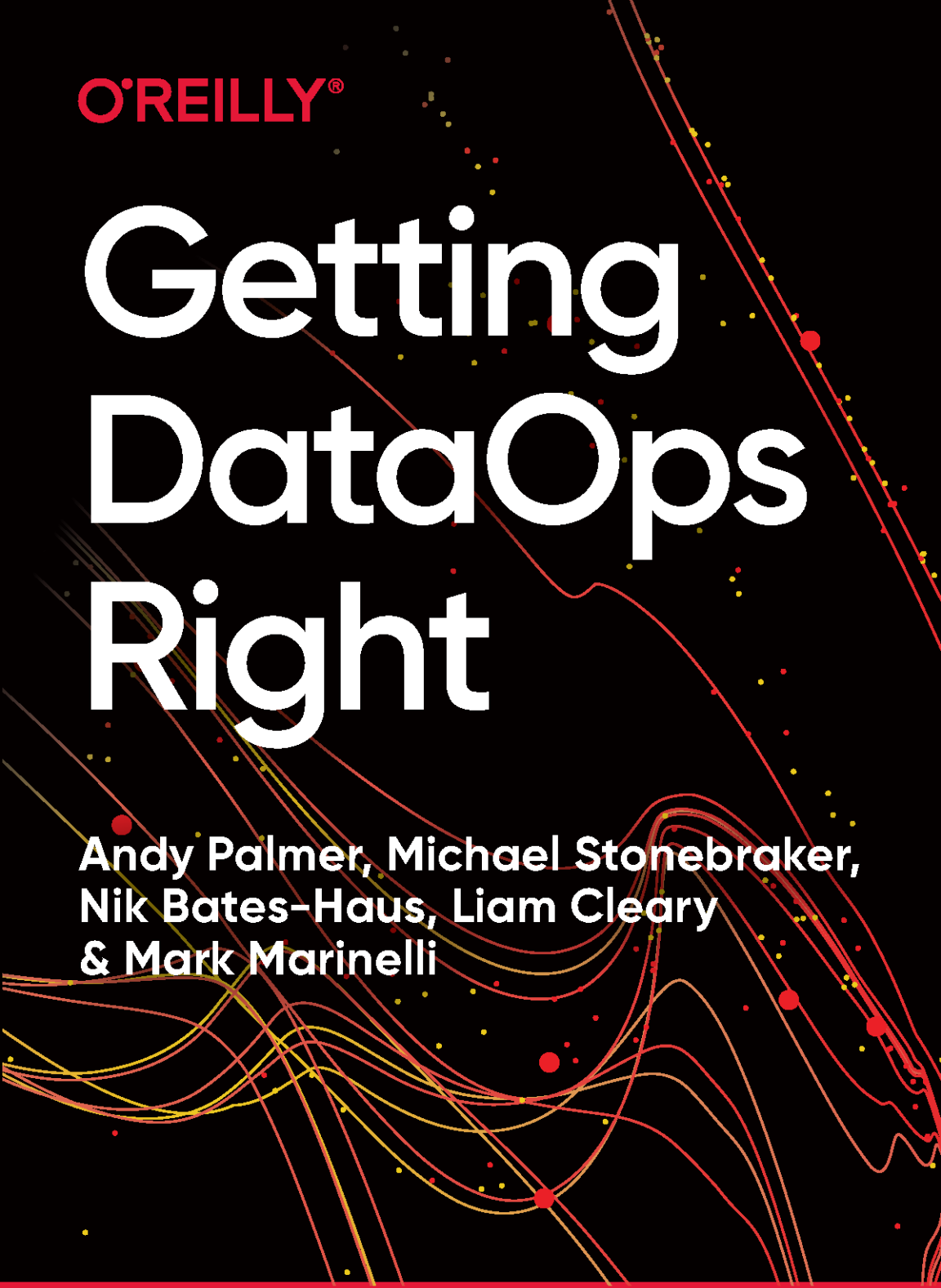
O'REILLY®
Getting DataOps Right
Andy Palmer, Michael Stonebraker, Nik Bates-Haus, Liam Cleary & Mark Marinelli
REPORT

Getting DataOps Right

Andy Palmer, Michael Stonebraker,
Nik Bates-Haus, Liam Cleary,
and Mark Marinelli

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Getting DataOps Right
by Andy Palmer, Michael Stonebraker, Nik Bates-Haus, Liam Cleary, and Mark Marinelli

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Development Editor: Jeff Bleiel
Acquisitions Editor: Rachel Roumeliotis
Production Editor: Katherine Tozer
Copyeditor: Octal Publishing, Inc.
Proofreader: Charles Roumeliotis
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

June 2019: First Edition

Revision History for the First Edition
2019-07-01: First Release

This work is part of a collaboration between O'Reilly and Tamr. See our statement of editorial independence.

978-1-492-03175-8

[LSI]

Table of Contents

CHAPTER 1
Introduction

Andy Palmer

Over the past three decades, as an enterprise CIO and a provider of third-party enterprise software, I've witnessed firsthand a long series of large-scale information technology transformations, including client/server, Web 1.0, Web 2.0, the cloud, and Big Data. One of the most important but underappreciated of these transformations is the astonishing emergence of DevOps.

DevOps—the ultimate pragmatic evolution of Agile methods—has enabled digital-native companies (Amazon, Google, etc.) to devour entire industries through rapid feature velocity and rapid pace of change, and is one of the key tools being used to realize Marc Andreessen's portent that "Software Is Eating the World". Traditional enterprises, intent on competing with digital-native internet companies, have already begun to adopt DevOps at scale. While running software and data engineering at the Novartis Institute of Biomedical Research, I introduced DevOps into the organization, and the impact was dramatic.

Fundamental changes such as the adoption of DevOps tend to be embraced by large enterprises after new technologies have matured to a point when the benefits are broadly understood, the cost and lock-in of legacy/incumbent enterprise vendors becomes insufferable, and core standards emerge through a critical mass of adoption. We are witnessing the beginning of another fundamental change in enterprise tech called "DataOps"—which will allow enterprises to rapidly and repeatedly engineer mission-ready data from all of the data sources across an enterprise.

DevOps and DataOps

Much like DevOps in the enterprise, the emergence of enterprise DataOps mimics the practices of modern data management at large internet companies over the past 10 years. Employees of large internet companies use their company's data as a company asset, and leaders in traditional companies have recently developed this same appetite to take advantage of data to compete. But most large enterprises are unprepared, often because of behavioral norms (like territorial data hoarding) and because they lag in their technical capabilities (often stuck with cumbersome extract, transform, and load [ETL] and master data management [MDM] systems). The necessity of DataOps has emerged as individuals in large traditional enterprises realize that they should be using all the data generated in their company as a strategic asset to make better decisions every day. Ultimately, DataOps is as much about changing people's relationship to data as it is about technology infrastructure and process.

The engineering framework that DevOps created is great preparation for DataOps. For most enterprises, many of whom have adopted some form of DevOps for their IT teams, the delivery of high-quality, comprehensive, and trusted analytics using data across many data silos will allow them to move quickly to compete over the next 20 years or more. Just like the internet companies needed DevOps to provide a high-quality, consistent framework for feature development, enterprises need a high-quality, consistent framework for rapid data engineering and analytic development.

The Catalyst for DataOps: "Data Debt"

DataOps is the logical consequence of three key trends in the enterprise:

- Multibillion-dollar business process automation initiatives over the past 30-plus years that started with back-office system automation (accounting, finance, manufacturing, etc.) and swept through the front office (sales, marketing, etc.) in the 1990s and 2000s, creating hundreds, even thousands, of data silos within large enterprises.

- The competitive pressure of digital-native companies in traditional industries.
- The opportunity presented by the "democratization of analytics" driven by new products and companies that enabled broad use of analytic/visualization tools such as Spotfire, Tableau, and BusinessObjects.

For traditional Global 2000 enterprises intent on competing with digital natives, these trends have combined to create a major gap between the intensifying demand for analytics among empowered frontline people and the organization's ability to manage the "data exhaust" from all the silos created by business process automation.

Bridging this gap has been promised before, starting with data warehousing in the 1990s, data lakes in the 2000s, and decades of other data integration promises from the large enterprise tech vendors. Despite the promises of single-vendor data hegemony by the likes of SAP, Oracle, Teradata, and IBM, most large enterprises still face the grim reality of intensely fractured data environments. The cost of the resulting data heterogeneity is what we call "data debt."

Data debt stems naturally from the way that companies do business. Lines of businesses want control and rapid access to their mission-critical data, so they procure their own applications, creating data silos. Managers move talented personnel from project to project, so the data systems owners turn over often. The high historical rate of failure for business intelligence and analytics projects makes companies rightfully wary of game-changing and "boil the ocean" projects that were epitomized by MDM in the 1990s.

Paying Down the Data Debt

Data debt is often acquired by companies when they are running their business as a loosely connected portfolio, with the lines of business making "free rider" decisions about data management. When companies try to create leverage and synergy across their businesses, they recognize their data debt problem and work overtime to fix it. We've passed a tipping point at which large companies can no longer treat the management of their data as optional based on the whims of line-of-business managers and their willingness to fund central data initiatives. Instead, it's finally time for enterprises to tackle their data debt as a strategic competitive imperative. As my

friend Tom Davenport describes in his book *Competing on Analytics*, those organizations that are able to make better decisions faster are going to survive and thrive. Great decision making and analytics requires great unified data—the central solution to the classic garbage in/garbage out problem.

For organizations that recognize the severity of their data debt problem and determine to tackle it as a strategic imperative, DataOps enables them to pay down their data debt by rapidly and continuously delivering high-quality, unified data at scale from a wide variety of enterprise data sources.

From Data Debt to Data Asset

By building their data infrastructure from scratch with legions of talented engineers, digital-native, data-driven companies like Facebook, Amazon, Netflix, and Google have avoided data debt by managing their data as an asset from day one. Their examples of treating data as a competitive asset have provided a model for savvy leaders at traditional companies who are taking on digital transformation while dealing with massive legacy data debt. These leaders now understand that managing their data proactively as an asset is the first, foundational step for their digital transformation—it cannot be a "nice to have" driven by corporate IT. Even for managers who aren't excited by the possibility of competing with data, the threat of a traditional competitor using their data more effectively or disruption from data-driven, digital-native upstarts requires that they take proactive steps and begin managing their data seriously.

DataOps to Drive Repeatability and Value

Most enterprises have the capability to find, shape, and deploy data for any given idiosyncratic use case, and there is an abundance of analyst-oriented tools for "wrangling" data from great companies such as Trifacta and Alteryx. Many of the industry-leading executives I work with have commissioned and benefitted from one-and-done analytics or data integration projects. These idiosyncratic approaches to managing data are necessary but not sufficient to solve their broader data debt problem and to enable these companies to compete on analytics.

Next-level leaders who recognize the threat of digital natives are looking to use data aggressively and iteratively to create new value every day as new data becomes available. The biggest challenge faced in enterprise data is repeatability and scale—being able to find, shape, and deploy data reliably with confidence. Also—much like unstructured content on the web—structured data changes over time. The right implementation of DataOps enables your analytics to adapt and change as more data becomes available and existing data is enhanced.

Organizing by Logical Entity

DataOps is the framework that will allow these enterprises to begin their journey toward treating their data as an asset and paying down their data debt. The human behavioral changes and process changes that are required are as important, if not more important, than any bright, shiny new technology. In the best projects I've been involved with, the participants realize that their first goal is to organize their data along their key, logical business entities, examples of which include the following:

- Customers
- Suppliers
- Products
- Research
- Facilities
- Employees
- Parts

Of course, every enterprise and industry has its own collection of key entities. Banks might be interested in entities that allow fraud detection; agricultural firms might care more about climate and crop data. But for every enterprise, understanding these logical entities across many sources of data is key to ensuring reliable analytics. Many DataOps projects begin with a single entity for a single use case and then expand; this approach connects the data engineering activities to ROI from either selling more products or saving money through using unified, clean data for a given entity for analytics and decision making.

For each of these key entities, any chief data officer should be able to answer the following fundamental questions:

- What data do we have?
- Where does our data come from?
- Where is our data consumed?

To ensure clean, unified data for these core entities, a key component of DataOps infrastructure is to create a system of reference that maps a company's data to core logical entities. This unified system of reference should consist of unified attributes constructed from the raw physical attributes across source systems. Managing the pathways between raw, physical attributes, changes to the underlying data, and common operations on that data to shape it into production-readiness for the authoritative system of reference are the core capabilities of DataOps technologies and processes.

This book gets into much more detail on DataOps and the practical steps enterprises have and should take to pay down their own data debt—including behavioral, process, and technology changes. It traces the development of DataOps and its roots in DevOps, best practices in building a DataOps ecosystems, and real-world examples. I'm excited to be a part of this generational change, one that I truly believe will be a key to success for enterprises over the next decade as they strive to compete with their new digital-native competitors.

The challenge for the large enterprise with DataOps is that if it doesn't adopt this new capability quickly, it runs the risk of being left in the proverbial competitive dust.

CHAPTER 2

Moving Toward Scalable Data Unification

Michael Stonebraker

The early users of data management systems performed business data processing—mostly transactions (updates) and queries on the underlying datasets. These early applications enabled analytics on the current state of the enterprise. About two decades ago enterprises began keeping historical transactional data in what came to be called *data warehouses*. Such systems enabled the use of analytics to find trends over time; for example, pet rocks are out and Barbie dolls are in. Every large enterprise now has a data warehouse, on which business analysts run queries to find useful information.

The concept has been so successful that enterprises typically now have several-to-many analytical data stores. To perform cross-selling, obtain a single view of a customer, or find the best pricing from many supplier data stores, it is necessary to perform data unification across a collection of independently constructed data stores.

This chapter discusses the history of data unification and current issues.

A Brief History of Data Unification Systems

ETL systems were used early on to integrate data stores. Given the required amount of effort by a skilled programmer, ETL systems typically unified only a handful of data stores, fewer than two dozen in most cases. The bottleneck in these systems was the human time

required to transform the data into a common format for the destination repository—it was necessary to write "merge rules" to combine the data sources, and additional rules to decide on the true value for each attribute in each entity. Although fine for small operations, like understanding sales and production data at a handful of retail stores or factories, ETL systems failed to scale to large numbers of data stores and/or large numbers of records per store.

The next generation of ETL tools offered increased functionality, such as data cleaning capabilities and adaptors for particular data sources. Like the first generation, these ETL tools were designed for use by computer programmers, who had specialized knowledge. Hence, they did not solve the fundamental scalability bottleneck: the time of a skilled software professional. These ETL tools form the bulk of the unification market today; however, most large enterprises still struggle to curate data from more than a couple dozen sources for any given data unification project. The present state of affairs is an increasing number of data sources that enterprises want to unify and a collection of traditional ETL tools that do not scale. The rest of this book discusses scalability issues in more detail.

Unifying Data

The benefits to unifying data sources are obvious. If a category manager at Airbus wants to get the best terms for a part that their line of business (LoB) is buying, that manager will typically have access only to purchasing data from their own LoB. The ability to see what other LoBs are paying for a given part can help that category manager optimize their spend. Added up across all of the parts and suppliers across all Airbus LoBs, these insights represent significant savings. However, that requires integrating the LoB supplier databases for each LoB. For example, GE has 75 of them, and many large enterprises have several to many of them because every acquisition comes with its own legacy purchasing system. Hence, data unification must be performed at scale, and ETL systems are not up to the challenge.

The best approach to integrating two data sources of 20 records each is probably a whiteboard or paper and pencil. The best approach for integrating 20 data sources of 20,000 records each might very well be an ETL system and rules-based integration approach. However, if GE wants to unify 75 data sources with 10 million total records,

neither approach is likely to be successful. A more scalable strategy is required.

Unfortunately, enterprises are typically operating at a large scale, with orders of magnitude more data than ETL tools can manage. Everything from accounting software to factory applications are producing data that yields valuable operational insight to analysts working to improve enterprise efficiency. The easy availability and value of data sources on the web compounds the scalability challenge.

Moreover, enterprises are not static. For example, even if Airbus had unified all of its purchasing data, its acquisition of Bombardier adds the data of another enterprise to the unification problem. Scalable data unification systems must accommodate the reality of shifting data environments.

Let's go over the core requirements for unifying data sources. There are seven required processes:

1. Extracting data from a data source into a central processing location
2. Transforming data elements (e.g., WA to Washington)
3. Cleaning data (e.g., –99 actually means a null value)
4. Mapping schema to align attributes across source datasets (e.g., your "surname" is my "Last_Name")
5. Consolidating entities, or clustering all records thought to represent the same entity (e.g., are Ronald McDonald and R. MacDonald the same clown?)
6. Selecting the "golden value" for each attribute for each clustered entity
7. Exporting unified data to a destination repository

Plainly, requirements 2 through 5 are all complicated by scale issues. As the number and variety of data sources grows, the number and variety of required transforms and cleaning routines will increase commensurately, as will the number of attributes and records that need to be processed. Consider, for example, names for a given attribute, phone number, as shown here:

Source	Attribute name	Record format
CRM-1	Tel.	(xxx) xxx-xxxx
CRM-2	Phone_Number	Xxxxxxxxx
DataLake	Phone_Number	xxx-xxx-xxxx

To consolidate the two CRM sources into the DataLake schema, you need to write one mapping: Phone_Number equals Telephone. To standardize the format of the number, you need to transform two different formats to a third standard one.

Now let's do this for six data sources:

Source	Attribute name	Record format
CRM-1	Tel.	(xxx) xxx-xxxx
CRM-2	Phone_Number	Xxxxxxxxx
Excel-1	Phone	(xxx) xxx-xxxx
Excel-2	Telephone Number	xxx.xxx.xxxx
POS-1	Cell	Xxx xxx xxxx
DataLake	Phone_Number	xxx-xxx-xxxx

We now have five different names for the same attributes, and one of these attributes (Cell) might require some expertise to correctly map it to Phone_Number. We also have four formats for phone numbers, requiring four different transformations into the DataLake format. In this simple example, we've gone from three rules to unify three data sources to eight when doubling the amount of attributes. Hence, the complexity of the problem is increasing much faster than the number of data sources. Rules are problematic at scale because:

- They are difficult to construct.
- After a few hundred, they surpass the ability of a human to understand them.
- At scale, they outstrip the ability of humans to verify them.

The first and second generations of ETL systems relied on rules. Creating and maintaining rules, in additional to the verification of the results of those rules, constitutes the bulk of the human time required for rules-based ETL approaches. This is an example of why traditional ETL solutions do not scale. Any scalable data unification must obey the tenets discussed in the next section.

Rules for Scalable Data Unification

A scalable approach therefore must perform the vast majority of its operations automatically (*tenet 1*). Suppose that it would take Airbus 10 years of labor to integrate all of its purchasing systems using a traditional, rules-based approach. If we could achieve 95% automation, it would reduce the time scale of the problem to six months. Automation, in this case, would use statistics and machine learning to make automatic decisions wherever possible, and involve a human only when automatic decisions are not possible. In effect, we must reverse the traditional ETL architecture, whereby a human controls the processing, into one in which a computer runs the process using human help when necessary.

For many organizations, the large number of data sources translates into a substantial number of attributes; thousands of data sources can mean tens or hundreds of thousands of attributes. We know from experience that defining a global schema upfront, although tempting, inevitably fails, because these schemas are invalid as soon as requirements change or new data sources are added. Scalable data unification systems should be discovered from the source attributes themselves rather than defined first. Therefore, scalable data unification must be schema-last (*tenet 2*).

As mentioned earlier, ETL systems require computer programmers to do the majority of the work. Business experts are sometimes involved in specifying requirements, but the people who build and maintain the data architecture are also responsible for interpreting the data they are working with. This requires, for example, a data architect to know whether "Merck KGaA" is the same customer as "Merck and Co." Obviously, this requires a business expert. As a result, scalable data unification systems must be collaborative and use domain experts to resolve ambiguity, thereby assisting the computer professionals who run the unification pipeline (*tenet 3*).

Taken together, these three tenets lead us to a fourth one, which is rules-based systems will not scale, given the limitations outlined earlier. Only machine learning can scale to the problem sizes found in large enterprises (*tenet 4*).

However, machine learning–based solutions do have some operational complexities to consider. Although a human can look at a set of records and instantly decide that they correspond to a single entity, data unification systems must do so automatically. Conven-

tional wisdom is to cluster records into a multidimensional space formed by the records' attributes, with a heuristically specified distance function. Records that are close together in this space are probably the same entity. This runs into the classic *N***2 clustering problem, and the computational resource required to do operations with complexity *N***2, where *N* is the number of records, is often too great. Scalable unification systems must scale out to multiple cores and processors (*tenet 5*) and must have a parallel algorithm with lower complexity than *N***2 (*tenet 6*).

Given the realities of the enterprise data ecosystem, scalable unification systems need to accommodate data sources that change regularly. Even though running the entire workflow on all of the data to incorporate changes to a data source can satisfy some business use cases, applications with tighter latency requirements demand a scalable unification system to examine the changed records themselves and perform incremental unification (*tenet 7*).

Scalable data unification must be the goal of any enterprise, and that will not be accomplished using traditional ETL systems. It is obviously the foundational task for enterprises looking to gain "business intelligence gold" from across the enormous troughs of enterprise data.

CHAPTER 3
DataOps as a Discipline

Nik Bates-Haus

DataOps, like DevOps, emerges from the recognition that separating the product—production-ready data—from the process that delivers it—operations—impedes quality, timeliness, transparency, and agility. The need for DataOps comes about because data consumption has changed dramatically over the past decade. Just as internet applications raised user expectations for the usability, availability, and responsiveness of applications, things like Google Knowledge Panel and Wikipedia have dramatically raised user expectations for the usability, availability, and freshness of data.

What's more, with increased access to very usable self-service data preparation and visualization tools, there are also now many users within the enterprise who are ready and able to prepare data for their own use if official channels are unable to meet their expectations. In combination, these changes have created an environment in which continuing with the cost-laden, delay-plagued, opaque operations used to deliver data in the past are no longer acceptable. Taking a cue from DevOps, DataOps looks to combine the production and delivery of data into a single, Agile practice that directly supports specific business functions. The ultimate goal is to cost-effectively deliver timely, high-quality data that meets the ever-changing needs of the organization.

In this chapter, we review the history of DataOps, the problems it is designed to address, the tools and processes it uses, and how organizations can effectively make the transition to and gain the benefits of DataOps.

DataOps: Building Upon Agile

DataOps is a methodology that spans people, processes, tools, and services to enable enterprises to rapidly, repeatedly, and reliably deliver production data from a vast array of enterprise data sources to a vast array of enterprise data consumers.

DataOps builds on the many decades of accumulated wisdom received from Agile processes. It is worth taking a moment to highlight some key goals and tenets of Agile, how they have been applied to software, and how they can be applied to data. Agile software development arose from the observation that software projects that were run using traditional processes were plagued by the following:

- High cost of delivery, long time to delivery, and missed deadlines
- Poor quality, low user satisfaction, and failure to keep pace with ever-changing requirements
- Lack of transparency into progress toward goals as well as schedule unpredictability
- Anti-scaling in project size, where the cost per feature of large projects is higher than the cost per feature of small projects
- Anti-scaling in project duration, where the cost of maintenance grows to overwhelm available resources

These are the same frustrations that plague so many data delivery projects today.

The Agile Manifesto

In establishing an approach that seeks to address each of these issues, the Agile community introduced several core tenets in an Agile Manifesto:

We value:

1. *Individuals and interactions* over processes and tools
2. *Working software* over comprehensive documentation
3. *Customer collaboration* over contract negotiation
4. *Responding to change* over following a plan

That is, while there is value in the items on the right, we value the items on the left more. Let's review these briefly, their impact on software development, and the expected impact on data delivery.

Tenet 2: Working software

I'll start with tenet 2, because it really should be tenet 1: the goal of software engineering is to deliver working software. Everything else is secondary. With working software, users can accomplish their goals significantly more readily than they could without the software. This means that the software meets the users' functional needs, quality needs, availability needs, serviceability needs, and so on. Documentation alone doesn't enable users to accomplish their goals.

Similarly, the goal of data engineering is to produce working data; everything else is secondary. With working data, users can accomplish their goals significantly more readily than they could without the data. Ideally, data engineering teams will be able to adhere to principles of usability and data design that make documentation unnecessary for most situations.

The other three tenets are in support of this main tenet. They all apply equally well to a data engineering team, whose goal is to produce working data.

Tenet 1: Individuals and interactions

Software is written by people, not processes or tools. Good processes and tools can support people and help them be more effective, but neither processes nor tools can make mediocre engineers into great engineers. Conversely, poor processes or tools can reduce even the best engineers to mediocrity. The best way to get the most from your team is to support them as people, first, and to bring in tools and processes only as necessary to help them be more effective.

Tenet 3: Customer collaboration

When you try to capture your customers' needs up front in a requirements "contract," customers will push for a very conservative contract to minimize their risk. Building to this contract will be very expensive and still not likely meet customers' real needs. The best way to determine whether a product meets your customer's needs and expectations is to have the customer use the product and give

feedback. Getting input as early and as often as possible ensures course corrections are as small as possible.

Tenet 4: Responding to change

Change is constant—in requirements, in process, in availability of resources, and so on—and teams that fail to adapt to these changes will not deliver software that works. No matter how good a plan is, it cannot anticipate the changes that will happen during execution. Rather than invest heavily in upfront planning, it is much better to plan only as much as necessary to ensure that the team is aligned and the goals are reasonable and then measure often to determine whether a course correction is necessary. Only by adapting swiftly to change can the cost of adaptation be kept small.

Agile Practices

The preceding has described the goal and tenets of Agile, but not what to actually *do*. There are many variations of the Agile process, but they share several core recommendations:

Deliver working software frequently
: In days or weeks, not months or years, adding functionality incrementally until a release is completed.

Get daily feedback from customers (or customer representatives)
: Gather feedback on what has been done so far.

Accept changing requirements
: Be prepared to do so even late in development.

Work in small teams
: Work in teams of three to seven people who are motivated, trusted, and empowered individuals, with all the skills required for delivery present on each team.

Keep teams independent
: This means each team's responsibilities span all domains, including planning, analysis, design, coding, unit testing, acceptance testing, releasing, and building and maintaining tools and infrastructure.

Continually invest in automation
: You should aim to automate everything.

Continually invest in improvement
: Again, automate everything, including process, design, and tools.

These practices have enabled countless engineering teams to deliver timely, high-quality products, many of which we use every day. These same practices are now enabling data engineering teams to deliver the timely, high-quality data that powers applications and analytics. But there is another transition made in the software world that needs to be picked up in the data world. When delivering hosted applications and services, Agile software development is not enough. It does little good to rapidly develop a feature if it then takes weeks or months to deploy it, or if the application is unable to meet availability or other requirements due to inadequacy of the hosting platform. These are operations, and they require a skill set quite distinct from that of software development. The application of Agile to operations created DevOps, which exists to ensure that hosted applications and services not only can be developed, but also delivered in an Agile manner.

Agile Operations for Data and Software

Agile removed many barriers internal to the software development process and enabled teams to deliver production features in days, instead of years. For hosted applications in particular, the follow-on process of getting a feature deployed retained many of the same problems that Agile intended to address. Bringing development and operations into the same process, and often the same team, can reduce time-to-delivery down to hours or minutes. The principle has been extended to operations for nonhosted applications, as well, with similar effect. This is the core of DevOps. The problems that DevOps intends to address look very similar to those targeted by Agile software development:

- Improved deployment frequency
- Faster time to market
- Lower failure rate of new releases
- Shortened lead time between fixes
- Faster mean time to recovery (MTTR) (in the event of a new release crashing or otherwise disabling the current system)

We can summarize most of these as *availability*—making sure that the latest working software is consistently available for use. To determine whether a process or organization is improving availability, you need something more transparent than percent uptime, and you need to be able to measure it continuously so that you know when you're close, and when you're deviating. Google's Site Reliability Engineering team did some of the pioneering work looking at how to measure availability in this way, and distilled it into the measure of the fraction of requests that are successful. DevOps, then, has the goal of maximizing the fraction of requests that are successful, at minimum cost.

For an application or service, a request can be logging in, opening a page, performing a search, and so on. For data, a request can be a query, an update, a schema change, and so forth. These requests might come directly from users (for example, on an analysis team) or they could be made by applications or automated scripts. Data development produces high-quality data, whereas DataOps ensures that the data is consistently available, maximizing the fraction of requests that are successful.

DataOps Tenets

DataOps is an emerging field, whereas DevOps has been put into practice for many years now. We can use our depth of experience with DevOps to provide a guide for the developing practice of DataOps. There are many variations in DevOps, but they share a collection of core tenets:

1. Think services, not servers
2. Infrastructure as Code
3. Automate everything

Let's review these briefly, how they affect service availability, and the expected impact on data availability.

Tenet 1: Think services, not servers

When it comes to availability, there are many more options for making a service available than there are for making a server available. By abstracting services from servers, we open up possibilities such as replication, elasticity, failover, and more, each of which can enable a service to successfully handle requests under conditions where an

individual server would not be successful, for example, under a sudden surge in load, or requests that come from broad geographic distribution.

This should make it clear why it is so important to think of data availability not as database server availability, but as the availability of Data as a Service (DaaS). The goal of the data organization is not to deliver a database, or a data-powered application, but the data itself, in a usable form. In this model, data is typically not delivered in a single form factor, but simultaneously in multiple form factors to meet the needs of different clients: RESTful web services to meet the needs of service-oriented applications; streams to meet the need of real-time dashboards and operations; and bulk data in a data lake for offline analytic use cases. Each of these delivery forms can have independent service-level objectives (SLOs), and the DataOps organization can track performance relative to those objectives when delivering data.

Tenet 2: Infrastructure as Code

A service can't be highly available if responding to an issue in its infrastructure depends on having the person with the right knowledge or skills available. You can't increase the capacity of a service if the configuration of its services isn't captured anywhere other than in the currently running instances. And you can't trust that infrastructure will be correctly deployed if it requires a human to correctly execute a long sequence of steps. By capturing all the steps to configure and deploy infrastructure as code, not only can infrastructure changes be executed quickly and reliably by anyone on the team, but that code can be planned, tested, versioned, released, and otherwise take full advantage of the depth of experience we have with software development.

With Infrastructure as Code (IaC), deploying additional servers is a matter of running the appropriate code, dramatically reducing the time to deployment as well as the opportunity for human error. With proper versioning, if an issue is introduced in a new version of a deployment, we can roll back the deployment to a previous version while the issue is identified and addressed. To further minimize issues found in production, we can deploy infrastructure in staging and user acceptance testing (UAT) environments, with full confidence that redeploying in production will not bring any surprises.

Using IaC enables operations to be predictable, reliable, and repeatable.

From the DataOps perspective, this means that everything involved in delivering data must be embodied in code. Of course, this includes infrastructure such as hosts, networking, and storage, but, importantly, this also covers everything to do with data storage and movement, from provisioning databases, to deploying ETL servers and data-processing workflows, to setting up permissions, access control, and enforcement of data governance policy. Nothing can be done as a one-off; everything must be captured in code that is versioned, tested, and released. Only by rigorously following this policy will data operations be predictable, reliable, and repeatable.

Tenet 3: Automate everything

Many of the techniques available for keeping services available will not work if they require a human in the loop. When there is a surge in demand, service availability will drop if deploying a new server requires a human to click a button. Deploying the latest software to production will take longer if a human needs to run the deployment script. Rather, all of these processes need to be automated. This pervasive automation unlocks the original goal of making working software highly available to users. With pervasive automation, new features are automatically tested both for correctness and acceptance; the test automation infrastructure is itself tested automatically; deployment of new features to production is automated; scalability and recovery of deployed services is automated (and tested, of course); and it is all monitored, every step of the way. This is what enables a small DevOps team to effectively manage a large infrastructure, while still remaining responsive.

Automation is what enables schema changes to propagate quickly through the data ecosystem. It is what ensures that responses to compliance violations can be made in a timely, reliable, and sustainable way. It is what ensures that we can uphold data freshness guarantees. And it is what enables users to provide feedback on how the data does or could better suit their needs so that the process of rapid iteration can be supported. Automation is what enables a small DataOps team to effectively keep data available to the teams, applications, and services that depend on it.

DataOps Practices

The role of the operations team is to provide the applications, services, and other infrastructure used by the engineering teams to code, build, test, package, release, configure, deploy, monitor, govern, and gather feedback on their products and services. Thus, the operations team is necessarily interdisciplinary. Despite this breadth, there are concrete practices that apply across all these domains:

Apply Agile process
: Short time-to-delivery and responsiveness to change (along with everything that comes with those requirements) are mandatory for the DataOps team to effectively support any other Agile team.

Integrate with your customer
: The DataOps team has the advantage that the customers, the engineering teams they support, are in-house, and therefore readily available for daily interaction. Gather feedback at least daily. If it's possible for DataOps and data engineering to be colocated, that's even better.

Implement everything in code
: This means host configuration, network configuration, automation, gathering and publishing test results, service installation and startup, error handling, and so on. Everything needs to be code.

Apply software engineering best practices
: The full value of IaC is attained when that code is developed using the decades of accumulated wisdom we have in software engineering. This means using version control with branching and merging, automated regression testing of everything, clear code design and factoring, clear comments, and so on.

Maintain multiple environments
: Keep development, acceptance testing, and production environments separate. Never test in production, and never run production from development. Note that one of the production environments for DataOps is the development environment for the data engineers, and another is the production environment for the data engineers. The DataOps development environment is for the DataOps team to develop new features and capabilities.

Integrate the toolchains
: The different domains of operations require different collections of tools ("toolchains"). These toolchains need to work together for the team to be able to be efficient. Your data movement engine and your version control need to work together. Your host configuration and your monitoring need to work together. You will be maintaining multiple environments, but within each environment, everything needs to work together.

Test everything
: Never deploy data if it hasn't passed quality tests. Never deploy a service if it hasn't passed regression tests. Automated testing is what allows you to make changes quickly, having confidence that problems will be found early, long before they get to production.

These practices enable a small operations team to integrate tightly with data engineering teams so that they can work together to deliver the timely, high-quality data that powers applications and analytics.

DataOps Challenges

DataOps teams, particularly those working with Big Data, encounter some challenges that other operations teams do not.

Application Data Interface

When integrating software packages into a single product, software engineers take advantage of application programing interfaces (APIs), which specify a functional and nonfunctional contract. Software subsystems can be written to provide or consume an API, and can be independently verified using a stubbed implementation on the other side of the API. These independently developed subsystems then can be fit together and will interoperate thanks to the contractual clarity of the API. There is no such equivalent for data. What we would like is an *application data interface* (ADI), which specifies a structural and semantic model of data so that data providers and data consumers can be verified independently and then fit together and trusted to interoperate thanks to the contractual clarity of the ADI. There have been multiple attempts to standardize representation of data structure and semantics, but there is no widely accepted standard. In particular, the Data Definition Lan-

guage (DDL) subset of SQL specifies structure and constraints of data, but not semantics. There are other standards for representing data semantics, but none have seen broad adoption. Therefore, each organization needs to independently select and employ tools to represent and check data model and semantics.

Data Processing Architecture

There are two fundamental modes for data: snapshots, represented in tables, and transactions, represented in streams. The two support different use cases, and, unfortunately, they differ in every respect, from structure, to semantics, to queries, to tools and infrastructure. Data consumers want both. There are well-established methods of modeling the two in the data warehousing world, but with the ascendency of data lakes, we are having to discover new methods of supporting them. Fortunately, the data warehousing lessons and implementation patterns transfer relatively cleanly to the technologies and contexts of contemporary data lakes, but because there is not yet good built-in tool support, the DataOps team will be confronted with the challenge of assembling and configuring the various technologies to deliver data in these modes.

There are now multiple implementation patterns that purport to handle both snapshot and streaming use cases while enabling a DataOps team to synchronize the two to a certain degree. Prominent examples are the Lambda Architecture and Kappa Architecture. Vendor toolchains do not yet have first-class support for such implementation patterns, so it is the task of the DataOps team to determine which architecture will meet their organization's needs and to deploy and manage it.

Query Interface

Data is not usable without a query interface. A query interface is a type of API, so data consumers can be written and verified against an abstract interface and then run against any provider of that API. Unfortunately, most query interfaces are vendor or vendor/version specific, and the vendors provide only one implementation of the query interface, so much of the benefit of writing to an API is lost. SQL is an attempt to create a standard data query API, but there is enough variation between vendor implementations that only the simplest of queries are compatible across vendors, and attaining

good performance always requires use of vendor-specific language extensions.

Thus, even though we want to focus on DaaS, independent of any particular vendor platform, the current reality is that the vendor and version of most query interfaces must be transparent to end users, and becomes part of the published interface of the data infrastructure. This impedes upgrades, and makes it nearly impossible to change vendors.

This problem is compounded by the fact that different data consumers require different kinds of query interface to meet their needs. There are three very different modes of interacting with data, and the DataOps team needs to provide interfaces for all of them:

- A *REST* interface to find, fetch, and update individual or small groups of records
- A *batch query* interface that supports aggregation over large collections of data
- A *streaming* interface that supports real-time analytics and alerting

The infrastructure, technology, and design of systems to support each of these kinds of query interface is very different. Many vendors provide only one or two of them and leave much of the complexity of deployment up to the DataOps team. The DataOps team needs to take this into consideration when designing their overall data processing architecture.

Resource Intensive

Even moderate-scale data places significant demands on infrastructure, so provisioning is another DataOps challenge. DataOps needs to consider data storage, movement, query processing, provenance, and logging. Storage must be provisioned for multiple releases of data as well as for different environments. Compute must be provisioned intelligently, to keep data transfers within acceptable limits. Network must be provisioned to support the data transfers that cannot be avoided. Although provisioning to support resource-intensive loads is not unique to DataOps, the nature of data is such that DataOps teams will have very little runway relative to other

kinds of teams before they begin to run into difficult challenges and trade-offs.

Schema Change

Vendors change data with every release. Analysts require data changes for every new analytic or visualization. These modifications put schemas, and therefore ADIs, in a state of perpetual change. Each change might require adjustment to the entire depth of the associated data pipelines and applications. Managing the entire DataOps ecosystem as versioned, tested code, with clear separation between development and production environments, makes it possible to respond quickly to these changes, with confidence that problems will be caught quickly. Unfortunately, many tools still assume that schemas change slowly or not at all, and the DataOps team must implement responsiveness to schema change outside these tools. Good factoring of code to centralize schema definition is the only way to keep up with this rapid pace of change.

Governance

Regulations from both government and industry cover data access, retention, traceability, accountability, and more. DataOps must support these regulations and provide alerting, logging, provenance, and so on throughout the data-processing infrastructure. Data governance tools are rapidly maturing, but interoperability between governance tools and other data infrastructure is still a significant challenge. The DataOps team will need to bridge the gaps between these toolchains to provide the coverage required by regulation.

The Agile Data Organization

DataOps in conjunction with Agile data engineering builds a next-generation data engineering organization. The goal of DataOps is to extend the Agile process through the operational aspects of data delivery so that the entire organization is focused on timely delivery of working data. Analytics is a major consumer of data, and DataOps in the context of Agile analytics has received quite a bit of attention. Other consumers also substantially benefit from DataOps, including governance, operations, security, and so forth. By combining the engineering skills that are able to produce the data with the operations skills that are able to make it available, this team is able to

cost-effectively deliver timely, high-quality data that meets the ever-changing needs of the data-driven enterprise.

This cross-functional team will now be able to deliver several key capabilities to the enterprise:[1]

Source data inventory
: Data consumers need to know what raw material is available to work with. What are the datasets, and what attributes do they contain? On what schedule is the source updated? What governance policies are they subject to? Who is responsible for handling issues? All of these questions need to be answered by the source data inventory.

Data movement and shaping
: Data needs to get from the sources into the enriched, cleaned forms that are appropriate for operations. This requires connectivity, movement, and transformation. All of these operations need to be logged, and the full provenance of the resulting data needs to be recorded.

Logical models of unified data
: Operations need to run on data models of entities that are tied to the business and are well understood. These models need to be concrete enough to enable practical use, while maintaining flexibility to accommodate the continuous change in the available and needed data.

Unified data hub
: The hub is a central location where users can find, access, and curate data on key entities—suppliers, customers, products, and more—that powers the entire organization. The hub provides access to the most complete, curated, and up-to-date information on these entities, and also identifies the provenance, consumers, and owners of that information.

Feedback
: At time of use, data quality issues become extremely transparent, so capturing feedback at point of use is critical to enabling the highest quality data. Every data consumer needs a readily

1 For more on this, see "DataOps: Building a Next Generation Data Engineering Organization" by Andy Palmer and Liam Cleary.

accessible feedback mechanism, powered by the Unified Data Hub. This ensures that feedback can be incorporated reliably and in the timeliest manner.

Combining DataOps with your Agile data engineering organization will allow you to achieve the transformational analytic outcomes that are so often sought, but that so frequently stumble on outdated operational practices and processes. Quickly and reliably responding to the demands presented by the vast array of enterprise data sources and the vast array of consumption use cases will build your "company IQ." DataOps is the transformational change data engineering teams have been waiting for to fulfill their aspirations of enabling their business to gain analytic advantage through the use of clean, complete, current data.

CHAPTER 4

Key Principles of a DataOps Ecosystem

Andy Palmer

Having worked with dozens of Global 2000 customers on their data/analytics initiatives, I have seen a consistent pattern of key principles of a DataOps ecosystem that is in stark contrast to traditional "single vendor," "single platform" approaches that are advocated by vendors such as Palantir, Teradata, IBM, Oracle, and others. An open, best of breed approach is more difficult, but also much more effective in the medium and long term; it represents a winning strategy for a chief data officer, chief information officer, and CEO who believe in maximizing the reuse of quality data in the enterprise and avoids the oversimplified trap of writing a massive check to a single vendor with the belief that there will be "one throat to choke."

There are certain key principles of a DataOps ecosystem that we see at work every day in a large enterprise. A modern DataOps infrastructure/ecosystem should be and do the following:

- Highly automated
- Open
- Take advantage of best of breed tools
- Use Table(s) In/Table(s) Out protocols
- Have layered interfaces
- Track data lineage

- Feature deterministic, probabilistic, and humanistic data integration
- Combine both aggregated *and* federated methods of storage and access
- Process data in both batch and streaming modes

I've provided more detail/thoughts on each of these in this chapter.

Highly Automated

The scale and scope of data in the enterprise has surpassed the ability of bespoke human effort to catalog, move, and organize the data. Automating your data infrastructure and using the principles of highly engineered systems—design for operations, repeatability, automated testing, and release of data—is critical to keep up with the dramatic pace of change in enterprise data. The principles at work in automating the flow of data from sources to consumption are very similar to those that drove the automation of the software build, test, and release process in DevOps over the past 20 years. This is one of the key reasons we call the overall approach "DataOps."

Open

The best way to describe this is to talk about what it is not. The primary characteristic of a modern DataOps ecosystem is that it is *not* a single proprietary software artifact or even a small collection of artifacts from a single vendor. For decades, B2B software companies have been in the business of trying to get their customers "hooked" on a software artifact by building it into their infrastructure—only to find that the software artifact is inadequate in certain areas. But because the software was built with proprietary assumptions, it's impossible to augment or replace it with other software artifacts that represent "best of breed" for that function.

In the next phase of data management in the enterprise, it would be a waste of time for customers to "sell their data souls" to single vendors that promote proprietary platforms. The ecosystem in DataOps should resemble DevOps ecosystems in which there are many best of breed free and open source software (FOSS) and proprietary tools that are expected to interoperate via APIs. An open ecosystem

results in better software being adopted broadly—and offers the flexibility to replace, with minimal disruption to your business, those software vendors that don't produce better software.

Best of Breed

Closely related to having an open ecosystem is embracing technologies and tools that are best of breed—meaning that each key component of the system is built for purpose, providing a function that is the best available at a reasonable cost. As the tools and technology that the large internet companies built to manage their data goes mainstream, the enterprise has been flooded with a set of tools that are powerful, liberating (from the traditional proprietary enterprise data tools), and intimidating.

Selecting the right tools for your workload is difficult because of the massive heterogeneity of data in the enterprise and also because of the dysfunction of software sales and marketing organizations who all overpromote their own capabilities, i.e., extreme data software vendor hubris.

It all sounds the same on the surface, so the only way to really figure out what systems are capable of is to try them (or by taking the word of a proxy—a real customer who has worked with the vendor to deliver value from a production system). This is why people such as Mark Ramsey, previously of GSK, are a powerful example. Mark's attempt to build an ecosystem for more than 12 best-of-breed vendors and combine their solutions to manage data as an asset at scale is truly unique and a good reference as to what works and what does not.

Table(s) In/Table(s) Out Protocol

The next logical questions to ask if you embrace best of breed is, "How will these various systems/tools communicate? And what is the protocol?" Over the past 20 years, I've come to believe when talking about interfaces between core systems that it's best to focus on the lowest common denominator. In the case of data, this means tables—both individual tables and collections.

I believe that Table(s) In/Table(s) Out is the primary method that should be assumed when integrating these various best of breed tools and software artifacts. Tables can be shared or moved using

many different methods described under Data Services. A great reference for these table-oriented methods is the popularity of Resilient Distributed Datasets (RDDs) and DataFrames in the Spark ecosystem. Using *service-oriented* methods for these interfaces is critical, and the thoughtful design of these services is a core component of a functional DataOps ecosystem. Overall, we see a pattern of three key types of interfaces that are required or desired inside of these systems.

Three Core Styles of Interfaces for Components

There are many personas that desire to use data in a large enterprise. Some "power users" need to access data in its raw form, whereas others just want to get responses to inquiries that are well formulated. A layered set of services and design patterns is required to satisfy all of these users over time.

Here are the three methods that we think are most useful (see also Figure 4-1):

- *Data access services* that are "View" abstractions over the data and are essentially SQL or SQL-like interfaces. This is the power-user level that data scientists prefer.
- *Messaging services* that provide the foundation for stateful data interchange, event processing, and data interchange orchestration.
- *REST services* built on or wrapped around APIs providing the ultimate flexible direct access to and interchange of data.

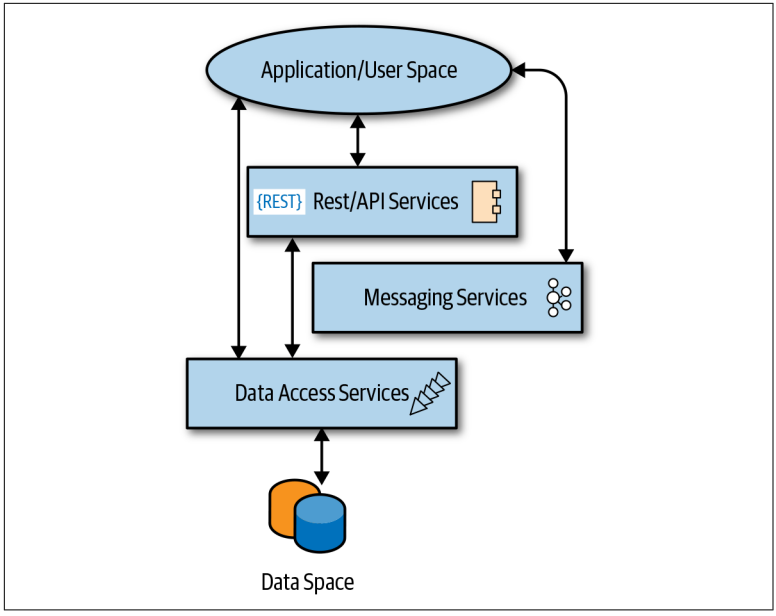

Figure 4-1. The layered set of data access services, messaging services, and REST services

Tracking Data Lineage and Provenance

As data flows through a next-generation data ecosystem (see Figure 4-2), it is of paramount importance to properly manage this lineage metadata to ensure reproducible data production for analytics and machine learning. Having as much provenance/lineage for data as possible enables reproducibility that is essential for any significant scale in data science practices or teams. Ideally, each version of a tabular input and output to a processing step is registered. In addition to tracking inputs and outputs to a data processing step, some metadata about what that processing step is doing is also essential. A focus on data lineage and processing tracking across the data ecosystem results in reproducibility going up and confidence in data increasing. It's important to note that lineage/provenance is not absolute—there are many subtle levels of provenance and lineages, and it's important to embrace the spectrum and appropriate implementation (i.e., it's more a style of your ecosystem than a component).

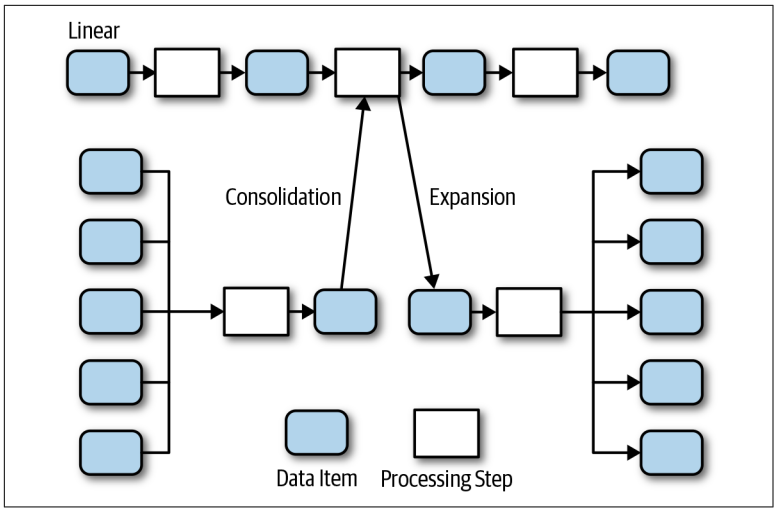

Figure 4-2. Data pipeline patterns

Data Integration: Deterministic, Probabilistic, and Humanistic

When bringing data together from disparate silos, it's tempting to rely on traditional deterministic approaches to engineer the alignment of data with rules and ETL. I believe that at scale—with many hundreds of sources—the only viable method of bringing data together is the use of machine-based models (probabilistic) + rules (deterministic) + human feedback (humanistic) to bind the schema and records together as appropriate in the context of both how the data is generated and (perhaps more importantly) how the data is being consumed.

Combining Aggregated and Federated Storage

A healthy next-generation data ecosystem embraces data that is both aggregated *and* federated. Over the past 40-plus years, the industry has gone back and forth between federated *or* aggregated approaches to integrating data. It's my strong belief that what is required in the modern enterprise is an overall architecture that embraces the idea that sources and intermediate storage of data will be a combination of both aggregated and federated data.

This adds a layer of complexity that was previously challenging, but actually possible now with some modern design patterns. You

always make trade-offs of performance and control when you aggregate versus federate. But over and over, I find that workloads across an enterprise (when considered broadly and holistically) require both aggregated and federated. In your modern DataOps ecosystem, cloud storage methods can make this much easier. In fact, Amazon Simple Storage Service (Amazon S3) and Google Cloud Storage (GCS) specifically—when correctly configured as a primary storage mechanism—can give you the benefit of both aggregated and federated methods.

Processing Data in Both Batch and Streaming Modes

The success of Kafka and similar design patterns has validated that a healthy next-generation data ecosystem includes the ability to simultaneously process data from source to consumption in *both* batch and streaming modes. With all the usual caveats about consistency, these design patterns can give you the best of both worlds—the ability to process batches of data as required and also to process streams of data that provide more real-time consumption.

Conclusion

Everything presented in this chapter is obviously high level and has an infinite number of technical caveats. After doing hundreds of implementations at large and small companies, I believe that it's actually possible to do all of it within an enterprise, but not without embracing an open and best-of-breed approach. At Tamr, we're in the middle of exercising all of these principles with our customers every day, across a diverse and compelling set of data engineering projects.

CHAPTER 5

Key Components of a DataOps Ecosystem

Andy Palmer

There are many ways to think about the potential components of a next-generation data ecosystem for the enterprise. Our friends at DataKitchen have done a good job with this post, which refers to some solid work by the Eckerson Group. In the interest of trying to simplify the context of what you might consider buying versus building and which vendors you might consider, I've tried to lay out the primary components of a next-generation enterprise data ecosystem based on the environments I've seen people configuring over the past 8 to 10 years and the tools (new and old) that are available. We can summarize these components as follows:

- Catalog/Registry
- Movement/ETL
- Alignment/Unification
- Storage
- Publishing
- Feedback

I provide a brief summary of each of these components in the sections that follow.

Catalog/Registry

Over the past 5 to 10 years, a key function has emerged as a critical starting point in the development of a functional DataOps ecosystem—the data catalog/registry. There are a number of FOSS and commercial projects that are attempts to provide tools that enable large enterprises to answer the simple question, "What data do you have?" Apache Atlas, Alation, and Waterline are the projects that I see the most in my work at Tamr and discussions with my chief data officer friends. I've always believed that the best data catalog/registry is a "vendor neutral" system that crawls all tabular data and registers all tabular datasets (files and inside of database management systems) using a combination of automation and human guidance.

The problem with vendor-specific systems is inevitably their implementation devolves into wars between vendor-oriented camps within your organization and you end up not with one catalog/registry, but multiple—which sort of defeats the purpose. If the vendor providing the catalog system is more open and interoperable (per Chapter 4 on the principles of a DataOps ecosystem), it's likely OK. This is not an impossible thing to build yourself—I've seen two to three built in the past few years that function well—but it is probably just a matter of time before something like Apache Atlas evolves to provide the basics in vendor-neutral systems that are highly open and interoperable.

Movement/ETL

There are *so* many options for data movement that it's a bit mind numbing. They range from the existing ETL/extract, load, and transform (ELT) vendors (Informatica, Talend, Oracle, IBM, Microsoft) to the new breed of movement vendors (my favorites are StreamSets, Data Kitchen, and KNIME) and the cloud data platform vendors such as Google GCP/DataFlow, Microsoft Azure, Amazon Web Services (AWS), Databricks, Snowflake, and the dozens of new ones that every VC in the Bay Area is funding by the month.

The most interesting dynamic here, in my opinion, is that most large enterprises use Python for most of their data pipelining, and the idea of broadly mandating a single pipelining tool for their users seems like a bit of a reach. Python is so easy for people to learn, and if you use an orchestration framework such as Apache AirFlow, you

get significant control and scale without all the overhead and restrictions of a proprietary framework. If you need massive ingest performance and scale, I think that something like StreamSets is your best bet, and I have seen this work incredibly well at GSK. However, most large companies will have requirements and heterogeneity among their pipelines that makes Python and AirFlow a better fit as the lowest common denominator across their enterprise.

One of the benefits of having a more open, interoperable, and best-of-breed approach is that as you need high-performance movement tools, you can adopt these incrementally. For example, you can use Python and Airflow as the baseline or default across your organization and then titrate in high-performance tools like StreamSets as required where you need the scale and performance. In the long term, this enables your ecosystem of tools to evolve gracefully and avoid massing single vendor lift-and-shift projects, which are prone to failure, despite the expectations that any single vendor might want to set along with an eight-figure-plus proposal.

Alignment/Unification

The tools required to create consistency in data need to be strongly rooted in the use of three key methods: rules, models, and human feedback. These three key methods, implemented with an eye toward an Agile process, are essential to tame the large challenge of variety in enterprise data. Traditional tools that depend on a single data architect to *a priori* define static schema, controlled vocabulary, taxonomy, ontologies, and relationships are inadequate to solve the challenges of data variety in the modern enterprise. The thoughtful engineering of data pipelines and the integration of rules, active learning–based probabilistic models of how data fits together, and the deliberate human feedback to provide subject matter expertise and handle corner cases is essential to success in the long-term alignment and molding of data broadly.

Storage

The biggest change in data systems over the past 10 years has been the evolution of data storage platforms—both cloud and on-premises. When my partner Mike and I started Vertica back in 2004, the database industry had plateaued in terms of new design patterns. At the time, everyone in the enterprise was using traditional "row

stores" for every type of work load—regardless of the fundamental fit. This was the gist of Mike and Ugur Cetintemel's "'One Size Fits All': An Idea Whose Time Has Come and Gone" paper and my Usenix talk in 2010. Overall, the key next-generation platforms that are top of mind now include the following:

- AWS: Redshift, Aurora, et al.
- GCP: BigTable, Spanner
- Azure: SQL Services et al.
- Databricks
- SnowFlake
- Vertica
- Postgres

The capabilities available in these platforms are dramatic and the pace of improvement is truly exceptional. I'm specifically not putting the traditional big vendors on this list—IBM, Oracle, Teradata—because most of them are falling further behind by the day. The cloud platform vendors have a *huge* advantage relative to the on-premises vendors in that the cloud vendors can radically improve their systems quickly without the latency associated with slow or long on-premises release cycles and the proverbial game of "telephone" that on-premises customers and vendors play—a game where it takes at least quarters and more often years and decades to get improvements identified, prioritized, engineered, and delivered into production.

The pace of change is what I believe will make the cloud-oriented platform vendors more successful than the on-premises platforms in the medium term. The best customers who are doing on-premises now are configuring their infrastructures to be compatible with cloud platforms so that when they migrate, they will be able to do so with minimal change. Those smart, on-premises vendors will have the advantage of a healthy abstraction from cloud platform vendors' proprietary services that could "lock in" other customers who start with that cloud platform from scratch.

Publishing

When data is organized and cleaned, providing a mechanism to broadly publish high-quality data is essential. This component delivers both a machine and human-readable form of dynamic datasets that have been broadly and consistently prepared. This component also has methods to recommend new data (rows, columns, values, relationships) that are discovered bottom up in the data over time. These methods can also be instrumented broadly into consumption endpoints such as analytic tools so that as new data becomes available, recommendations can be made dynamically to data consumers who might be interested in consuming the continuously improved and updated data in the context of their analytics, operational systems, and use.

Feedback

This is perhaps the most impactful and least appreciated component of a next-generation DataOps ecosystem. Currently, data in most enterprises flows unidirectionally from sources through deterministic and idiosyncratic pipelines toward data warehouses, marts, and spreadsheets—and that's where the story stops. There is an incredible lack of systematic feedback mechanisms to enable data to flow from where data is consumed back up into the pipelines and all the way back to the sources so that the data can be improved systematically over time. Most large organizations lack any "queue" of data problems identified by data consumers. At Tamr we've created "Steward" to help address this problem, providing a vendor-neutral queue of data consumer–reported problems with data for the enterprise.

Governance

Governance is a key component of a modern ecosystem—the evolution of data privacy has raised governance up to the top of the priority list—driven by a need to comply to many important new regulations. My belief is that the best governance infrastructure focuses on the automation of information access policy and the prosecution of that policy across users in the context of key roles that are aligned with policy. Focusing on governance in the context of information use helps avoid boiling the infinite proverbial ocean

of data source complexity. Having a codified information access policy as well as methods for running that policy in real time as users are consuming data should be the core goal of any governance infrastructure initiative.

CHAPTER 6

Building a DataOps Toolkit

Liam Cleary

The DataOps cross-functional toolkit is not a single tool or platform—nor can it ever be. Rather, it is part of a framework that prioritizes responding to change over following a plan. This means that the toolkit is inherently a collection of complementary, best of breed tools with interoperability and automation at their core design.

Interoperability

Interoperability is perhaps the biggest departure in the DataOps stack from the data integration tools or platforms of the past.

Although a good ETL platform certainly appeared to afford interoperability as a core principle, with many out-of-the-box connectors supporting hundreds of data formats and protocols (from Java Database Connectivity [JDBC] to Simple Object Access Protocol [SOAP]), this, in fact, was a reaction to the complete lack of interoperability in the set of traditional data repositories and tools.

In practice, if your ETL platform of choice did not support a native connector to a particular proprietary database, enterprise resource planning (ERP), customer relationship management (CRM), or business suite, the data was simply forfeited from any integration project. Conversely, data repositories that did not support an open data exchange format forced users to work within the confines of that data repository, often subsisting with rigid, suboptimal, worst-of-breed, tack-on solutions. The dream of being able to develop a center of excellence around a single data integration vendor quickly

evaporated as the rigid constraints of the single vendor tightened with increasing data variety and velocity.

Composable Agile Units

Just as an Agile development team comprises members covering each talent necessary for full feature development, so the Agile DataOps toolkit comprises each function necessary to deliver purpose-fit data in a timely manner. The Agile DataOps toolkit not only allows you to plug-and-play a particular tool between different vendor, open source or homegrown solutions, it also allows greater freedom in deciding the boundaries of your composable Agile units.

This is key when trying to compose Agile DataOps teams and tools while working within the realities of your available data engineering and operations skill sets. Whether your data dashboarding team member is a Tableau or Domo wizard does not determine which record matching engine you use. If your dashboarding team can work across many different record matching engines, then your record matching engine should be able to work across many different dashboarding tools. In stark contrast to the single platform approach, DataOps thrives in an environment of plug-and-play tools and capabilities. DataOps tools aspire to the mantra of doing one thing exceptionally well, and thus when taken together present a set of nonoverlapping complementary capabilities that align with your composable units.

Results Import

In practice, however, even DataOps tools can, and often must, overlap in capability. The answer to this apparent redundancy reveals one of the unique hallmarks of interoperability in the DataOps stack, namely the tool's ability to both export *and import* its results for common overlapping capabilities. In composing your DataOps pipeline, this allows your team to decide where best the artifacts and results of a particular common ability should be natively *generated* or *imported*.

Consider that a data extract tool, a data unification tool, and a data catalog are each capable of generating a basic dataset profile (record count, attribute percentage null, etc.):

- The data extract tool has direct connectivity to the raw sources and can use and extract physical properties and even metrics where available, producing a basic data quality profile.
- The data unification tool, although capable of generating a profile of the raw data it sees, might not have the same visibility as the extract tool and would benefit from being able to import the extract tool's profile. This same imported profile then sits alongside any generated profile, for example of a flat file to which the unification tool has direct access.
- The data cataloging tool, capable of profiling myriad file formats, catalogs both the raw datasets and the unified logical datasets, and benefits by importing both the extract tool dataset's profiles and the unification tool dataset's profiles, presenting the most informative view of the datasets.

Although each tool in the just-described DataOps toolkit is capable of generating a dataset profile, it is the ability of each tool to both *export and import* the dataset profile that is key. This ability allows the DataOps team great flexibility and control in composing a pipeline, designing around the core nonoverlapping ability of each component, while taking advantage of their import and export functionality to afford the interoperability of common abilities.

Metadata Exchange

The ability of the DataOps team to rapidly compose, build, and deploy data unification, cataloging, and analytics pipelines, with plug-and-play components, seems to present a formidable systems integration challenge. In DataOps however, the focus of interoperability is not the need for closer systems integration—for example, with each tool possessing native support for each other tool—but rather the need for open interoperable metadata exchange formats. This objective reveals the second unique hallmark of interoperability in the DataOps stack, namely the ready exchange and enrichment of metadata.

To illustrate, consider a data element richly decorated with the following metadata:

Data type
: `numeric`

To a data extraction tool, the data type is immediately valuable because knowing the data type might allow for faster extraction and removal of unnecessary type casting or type validation.

Description

`Total spend`

To a data unification tool, the description is immediately valuable because the description, when presented to an end user or algorithm, allows for more accurate mapping, matching, or categorization.

Format

`X,XXX.XX`

To a dashboarding tool, the data format is immediately valuable in presenting the raw data to an analyst in a meaningful, consumer-friendly manner.

To realize these benefits each of our DataOps tools requires the ability to both preserve (pass through) this metadata, and the ability to enrich it. For example, if we're lucky, the source system already contains this metadata on the data element. We now require our extraction tool to read, preserve, and pass this metadata to the unification tool. In turn, the unification tool preserves and passes this metadata to the cataloging tool. If we're unlucky and the source system is without this metadata, the extraction tool is capable of enriching the metadata by casting the element to `numeric`, the unification tool is capable of mapping to an existing data element with description `Total spend`, and the dashboarding tool applies a typical currency format `X,XXX.XX`.

In flowing from source, through extraction, to unification and dashboard, the metadata is preserved by each tool by supporting a common interoperable metadata exchange format. Equally important to preserving the metadata, each tool enriches the metadata and exposes it to each other tool. This interaction is in stark contrast to data integration practices espoused by schema-first methodologies. Although *metadata* pass-through and enrichment are certainly a secondary requirement to that of *data* pass-through and enrichment (which even still remains a challenge for many tools today), it is certainly a primary and distinguishing feature of a DataOps tool and is an essential capability for realizing interoperability in the DataOps stack.

Automation

One simple paradigm for automation is that every common UI action or set of actions, and any bulk or data-scale equivalent of these actions, is also available via a well-formed API. In the DataOps toolkit, this meeting of the API-first ethos of Agile development with the pragmatism of DevOps means that any DataOps tool should furnish a suite of APIs that allow the complete automation of its tasks.

Broadly speaking, we can consider that these tasks are performed as part of either *continuous* or *batch* automation. Continuous automation concerns itself with only a singular phase, namely that of updating or refreshing a preexisting state. In contrast, in batch automation, we encounter use cases that have a well-defined beginning (usually an empty or zero state), middle, and end, and automation is required explicitly around each of these phases. The suite of APIs that facilitate automation must concern itself equally with continuous and batch automation.

Continuous Automation

If the primary challenge of the DevOps team is to streamline the software release cycle to meet the demands of the Agile development process, so the objective of the DataOps team is to automate the continuous publishing of datasets and refreshing of every tool's results or view of those datasets.

To illustrate continuous automation in a DataOps project, let's consider a data dashboarding tool. We might ask the following questions of its API suite. Does the tool have the following:

- APIs that allow the updating of fundamental data objects such as datasets, attributes, and records?
- APIs to update the definition of the dashboard itself to use a newly-available attribute?
- APIs to update the dashboard's configuration?
- A concept of internally and externally created or owned objects? For example, how does it manage conflict between user and API updates of the same object?
- APIs for reporting health, state, versions, and up-to-dateness?

The ability of a tool to be automatically *updated and refreshed* is critical to delivering working data and meeting business-critical timeliness of data. The ease at which the dataset can be published and republished directly affects the feedback cycle with the data consumer and ultimately determines the DataOps team's ability to realize the goal of shortened lead time between fixes and faster MTTR.

Batch Automation

With most tools already providing sufficient capability to be continuously updated in an automatic, programmatic manner, it is easy to overlook the almost distinctly Agile need to spin up and tear down a tool or set of tools automatically.

Indeed, it is especially tempting to devalue batch automation as a once-off cost that need only be paid for a short duration at the very beginning when setting up a longer-term, continuous pipeline. However, the ability of a tool to be automatically initialized, started, run, and shut down is a core requirement in the DataOps stack and one that is highly prized by the DataOps team. Under the tenet of *automate everything*, it is essential to achieving responsiveness to change and short time to delivery.

To illustrate, let's consider batch automation features for a data unification tool. Does the tool have the following:

- APIs that allow the creation of fundamental data objects such as datasets, attributes, and records?
- APIs to import artifacts typically provided via the UI? For example, metadata about datasets, annotations and descriptions, configuration, and setup?
- APIs to perform complex user interactions or workflows? For example, manipulating and transforming, mapping, matching, or categorizing data?
- APIs for reporting backup, restore, and shutdown state? For example, can I ask the tool if shutdown is possible or if it must be forced?

A tool that can be automatically stood up from scratch and executed, where every step is codified, delivers on the goal of automate everything and unlocks the tenet of *test everything*. It is possible to not only spin up a complete, dataset-to-dashboard pipeline for

performing useful work (e.g., data model trials), but also to test the pipeline programmatically, running unit tests and data flow tests (e.g., did the metadata in the source dataset pass through successfully to the dashboard?). Batch automation is critical to realizing repeatability in your DataOps pipelines and a low error rate for newly published datasets.

As the key components of the DataOps ecosystem continue to evolve—meeting the demands of ever-increasing data variety and velocity and offering greater and greater capabilities for delivering working data—the individual tools that are deemed best of breed for these components will be those that prioritize interoperability and automation in their fundamental design.

CHAPTER 7

Embracing DataOps: How to Build a Team and Prepare for Future Trends

Mark Marinelli

So far, we've covered the catalyst behind and need for DataOps, the importance of an Agile mindset, principles and components of the DataOps ecosystem, and what to expect from a DataOps tool.

But for an organization looking to build a DataOps team, where do you start? And what are the major trends expected for the future that your organization should prepare for?

Building a DataOps Team

Technology is undeniably important, but people are also a vital cornerstone of the DataOps equation. A high-performance DataOps team rapidly produces new analytics and flexibly responds to marketplace demands. They unify the data from diverse, previously fragmented sources and transform it into a high-quality resource that creates value and enables users to gain actionable insights. A key aspect of embracing a DataOps mindset is for data engineering teams to begin thinking of themselves not as technicians who move data from source A to report B, but rather as software developers who employ Agile development practices to rapidly build data applications. This requires a combination of skills that might or might not already exist within your organization, and it requires an organ-

izational structure that formalizes some new functions and resources them accordingly.

We begin by identifying the key functions in the DataOps team. If you're versed in data management, the functions might sound familiar, but the DataOps methodology calls for different skill sets and working processes than have been traditionally employed, just as the DevOps approach to software development restructured existing development teams and tooling. The focus on agility and continuous iteration necessitates more collaboration across these functions to build and maintain a solid data foundation amid constantly shifting sources and demands.

Every large company buys lots of products and services to run their business, and all of them would like to improve the efficiency of their supplier relationships, so let's use supplier data as an example, looking at three of these functions.

Data Supply

Who owns your internal supplier management systems? Who owns your relationships with external providers or supplier data? The answer to these questions, typically found in the CIO's organization, is the data source supplier, of which you probably have dozens. As we transition from database views and SQL queries to data virtualization and APIs, from entity relationship diagrams (ERDs) and data dictionaries to schema-less stores and rich data catalogs, the expectations for discoverability and access to these sources have increased, but requirements for controlled access to sensitive data remain. In a DataOps world, these source owners must work together across departments and with data engineers to build the infrastructure necessary so that the rest of the business can take advantage of all data.

> A great data supplier can confidently say: "Here are *all* the sources that contain supplier data."

Data Preparation

End users of corporate data don't care about physical sources, they care about logical entities: in this case, suppliers. So, they need someone to prepare data for them that combines raw data from across all available sources, removes duplication, enriches with valuable information, and can be easily accessed. Effective data preparation requires a combination of technical skill to wrangle raw

sources and business-level understanding of how the data will be used. DataOps expands traditional preparation beyond the data engineers who move and transform data from raw sources to marts or lakes to also include the data stewards and curators responsible for both the quality and governance of critical data sources that are ready for analytics and other applications. The CDO is the ultimate executive owner of the data preparation function, ensuring that data consumers have access to high-quality, curated data.

> A great data preparation team can confidently say: "Here is *everything* we know about supplier X."

Data Consumption

On the "last mile" of the data supply chain, we have everyone responsible for using unified data for a variety of outcomes across analytical and operational functions. In our supplier data example, we have data analysts building dashboards charting aggregate spending with each supplier, data scientists building inventory optimization models, and data developers building Supplier 360 portal pages.

Modern visualization, analysis, and development tools have liberated these data consumers from some of the constraints of traditional business intelligence tools and data marts. However, they still must work closely with the teams responsible for providing them with current, clean, and comprehensive datasets. In a DataOps world, this means providing a feedback loop so that when data issues are encountered, they aren't merely corrected in a single dashboard but are instead communicated upstream so that actual root causes can be uncovered and (optimally) corrections can be made across the entire data community.

> A great data consumer can confidently say: "Here are our *actual* top 10 suppliers by annual spend, now and projected into next year."

So Where Are They?

Having defined the categories of roles we need, where do we find the right people? Often, this will require a combination of internal and external talent. And it's important to note that some of these roles are still evolving, so there are no job descriptions that can easily serve as guidelines.

Within the data supplier function, everyone's already in the building; you just need to be deliberate about nominating the right source owners and communicating to the rest of the organization who they are. A data catalog that has a single user who is accountable for providing access to each source and who should be the primary point of contact regarding issues with the quality of each source is a solid start.

Data preparation can be more of a challenge because it's rare when a single person combines technical skill with data management and a clear understanding of the business requirements of data. So, you'll need to seek out the most business-savvy members of your data engineering/ETL team and the most tech-savvy members of your business teams and pair them up. Stand up a project to build an analytical application to solve a well-bounded business problem, and get them started on assembling the necessary data pipelines in collaboration with the consumers of this data.

After you've identified the team that can build this pipeline, you now need to nominate an owner of the output of that preparation asset. Who is a primary stakeholder in the quality and governance of this new source? Who knows enough about its usage to improve quality and enforce governance? When you've found your answer, you've found a data steward. Add to this a person who should be responsible for maintaining information about this source in a data catalog, so that others know where to look for the best data to solve a particular problem, and you have a data curator as well.

Data consumption, like data supply, is already everywhere in your organization. A goal of DataOps is to ensure that these data consumers can concentrate on the analytical and operational problems that they want to solve, unencumbered by the difficult work of getting the useful data they need. You've hired some bright (and expensive) data scientists, and you'd like to hire more, but if they're spending the majority of their time wrangling data instead of building analytical models, no one is happy. With a robust DataOps data supply chain behind them, you can free up these skilled users to do the work for which you hired them. Not only will your data analysts or other users become more productive, but you can also expand the set of users who can actually use your data for transformational outcomes.

The good news: filling these roles is becoming easier. The skill set of the average data professional is increasing. More college students are learning data analytics and data science, and more traditional information workers are upskilling to include data analytics and data science in their repertoire.

The Future of DataOps

As the challenges associated with Big Data continue to increase along with enterprises' demands for easily accessible, unified data, what does the future of DataOps look like? In the following sections, I outline four important trends you should expect to see.

The Need for Smart, Automated Data Analysis

Internet of Things (IoT) devices will generate enormous volumes of data that must be analyzed if organizations want to gain insights—such as when crops need water or heavy equipment needs service. John Chambers, former CEO of Cisco, declared that there will be 500 billion connected devices by 2025. That's nearly 100 times the number of people on the planet. These devices are going to create a data tsunami.

People typically enter data into apps using keyboards, mice, or finger swipes. IoT devices have many more ways to communicate data. A typical mobile phone has nearly 14 sensors, including an accelerometer, GPS, and even a radiation detector. Industrial machines such as wind turbines and gene sequencers can easily have 100 sensors; a utility grid power sensor can send data 60 times per second and a construction forklift once per minute.

IoT devices are just one factor driving this massive increase in the amount of data enterprises can use. The end result of all of this new data is that its management and analysis will become more difficult and continue to strain or break traditional data management processes and tools. Only through increased automation via artificial intelligence (AI) and machine learning will this diverse and dynamic data be manageable economically.

Custom Solutions from Purpose-Built Components

The explosion of new types of data in great volumes has demolished the (erroneous) assumption that you can master Big Data through a

single platform (assuming that you'd even want to). The attraction of an integrated, single-vendor platform that turns dirty data into valuable information lies in its ability to avoid integration costs and risks.

The truth is, no one vendor can keep up with the ever-evolving landscape of tools to build enterprise data management pipelines and package the best ones into a unified solution. You end up with an assembly of second-tier approaches rather than a platform composed of best-of-breed components. When someone comes up with a better mousetrap, you won't be able to swap it in.

Many companies are buying applications designed specifically to acquire, organize, prepare, analyze, and visualize their own unique types of data. In the future, the need to stitch together purpose-built, interoperable technologies will become increasingly important for success with Big Data, and we will see some reference architectures coalesce, just as we've seen historically with LAMP, ELK, and others.

Organizations will need to turn to both open source and commercial components that can address the complexity of the modern data supply chain. These components will need to be integrated into end-to-end solutions, but fortunately they have been built to support the interoperability necessary to make them work together.

Increased Approachability of Advanced Tools

The next few years of data analysis will require a symbiotic relationship between human knowledge and technology. With more data in a variety of formats to deal with, organizations will need to take advantage of advancements in automation (AI and machine learning) to augment human talent.

Simultaneously, knowledge workers will need to improve their technical skills to bridge the gaps that technology cannot fill completely. Only through the combination of automation and increased human knowledge can organizations solve the problem of getting the right data to the right users so that they can make smarter, more beneficial decisions.

As more graduates who have studied data science and/or data engineering enter the workforce, and as existing knowledge workers upgrade their skills, the supply of data-proficient workers will increase. As more data management tools package AI/machine

learning in cleaner user interfaces, abstracting away the arcana of their underlying algorithms, we'll see the barriers to adoption of these newer techniques lower. These factors combine to form a powerful dynamic that will accelerate progress in the domain.

Subject Matter Experts Will Become Data Curators and Stewards

Organizations will need to think about crowdsourcing when it comes to data discoverability, maintenance, and quality improvement. The ultimate people required to make data unification truly effective are not data engineers, but rather highly contextual recommenders—subject matter experts—who, if directly engaged in the unification process, can enable a new level of productivity in data delivery.

Data consumers—nontechnical users—know customer, sales, HR, and other data by heart. They can assess the quality of the data and contribute their expertise to projects to improve data integrity. However, they are too busy to devote their time to the focused tasks of data curation and stewardship. There will be a huge opportunity for improvement as more people are allowed to work with the data that they know best and provide feedback on whether the data is accurate and valuable from within their existing tools and workflows.

Incorporating this data feedback systematically instead of having it locked up in emails or possibly never provided at all will produce dramatic gains in the ability to focus data quality efforts on the right problem sets, to correct issues with source data, and ultimately to prevent bad data from entering the enterprise in the first place.

A Final Word

Traditional data management techniques are adequate when datasets are static and relatively few. But they break down in environments of high volume and complexity. This is largely due to their top-down, rules-based approaches, which often require significant manual effort to build and maintain. These approaches are becoming extinct, quickly.

The future is inevitable—more data, technology advancements, and an increasing need for curation by subject matter experts. Data unification technology will help by connecting and mastering datasets

through the use of human-guided machine learning. The future is bright for organizations that embrace this new approach.

About the Authors

Andy Palmer is cofounder and CEO of Tamr, a data unification company, which he founded with fellow serial entrepreneur and 2014 Turing Award winner Michael Stonebraker, PhD, adjunct professor at MIT CSAIL; Ihab Ilyas, University of Waterloo; and others. Previously, Palmer was cofounder and founding CEO of Vertica Systems, a pioneering Big Data analytics company (acquired by HP). During his career as an entrepreneur, Palmer has served as founding investor, board of directors member, or advisor to more than 50 startup companies in technology, health care, and the life sciences. He also served as global head of software and data engineering at Novartis Institutes for BioMedical Research (NIBR) and as a member of the startup team and chief information and administrative officer at Infinity Pharmaceuticals. Additionally, he has held positions at Bowstreet, pcOrder.com, and Trilogy.

Michael Stonebraker is an adjunct professor at MIT CSAIL and a database pioneer who specializes in database management systems and data integration. He was awarded the 2014 A.M. Turing Award (known as the "Nobel Prize of computing") by the Association for Computing Machinery for his "fundamental contributions to the concepts and practices underlying modern database systems as well as their practical application through nine startup companies that he has founded."

Professor Stonebraker has been a pioneer of database research and technology for more than 40 years, and is the author of scores of papers in this area. Before joining CSAIL in 2001, he was a professor of computer science at the University of California Berkeley for 29 years. While at Berkeley, he was the main architect of the INGRES relational database management system (DBMS), the object-relational DBMS POSTGRES, and the federated data system Mariposa. After joining MIT, he was the principal architect of C-Store (a column store commercialized by Vertica), H-Store, a main memory OLTP engine (commercialized by VoltDB), and SciDB (an array engine commercialized by Paradigm4). In addition, he has started three other companies in the Big Data space, including Tamr, oriented toward scalable data integration. He also cofounded the Intel Science and Technology Center for Big Data, based at MIT CSAIL.

Nik Bates-Haus is a technology leader with more than two decades of experience building data engineering and machine learning technology for early-stage companies. Currently, he is a technical lead at Tamr, a machine learning–based data unification company, where he leads data engineering, machine learning, and implementation efforts. Prior to Tamr, he was director of engineering and lead architect at Endeca, where he was instrumental in the development of the search pioneer, which Oracle acquired for $1.1 billion. Previously, he delivered machine learning and data-integration platforms with Torrent Systems, Thinking Machines Corporation, and Philips Research North America. He has a master's degree in computer science from Columbia University.

Liam Cleary is a technical lead at Tamr, where he leads data engineering, machine learning, and implementation efforts. Prior to Tamr, he was a post-doctoral associate at MIT, researching quantum dissipative systems, before working as an internal consultant at Ab Initio, a data integration platform. He has a PhD in electrical engineering from Trinity College Dublin.

Mark Marinelli is a 20-year veteran of enterprise data management and analytics software. Currently, Mark is the Head of Product at Tamr, where he drives a product strategy that aligns innovation in data unification with evolving customer needs. Previously, Mark held roles in engineering, product management, and technology strategy at Lucent Technologies, Macrovision, and most recently Lavastorm, where he was Chief Technology Officer.