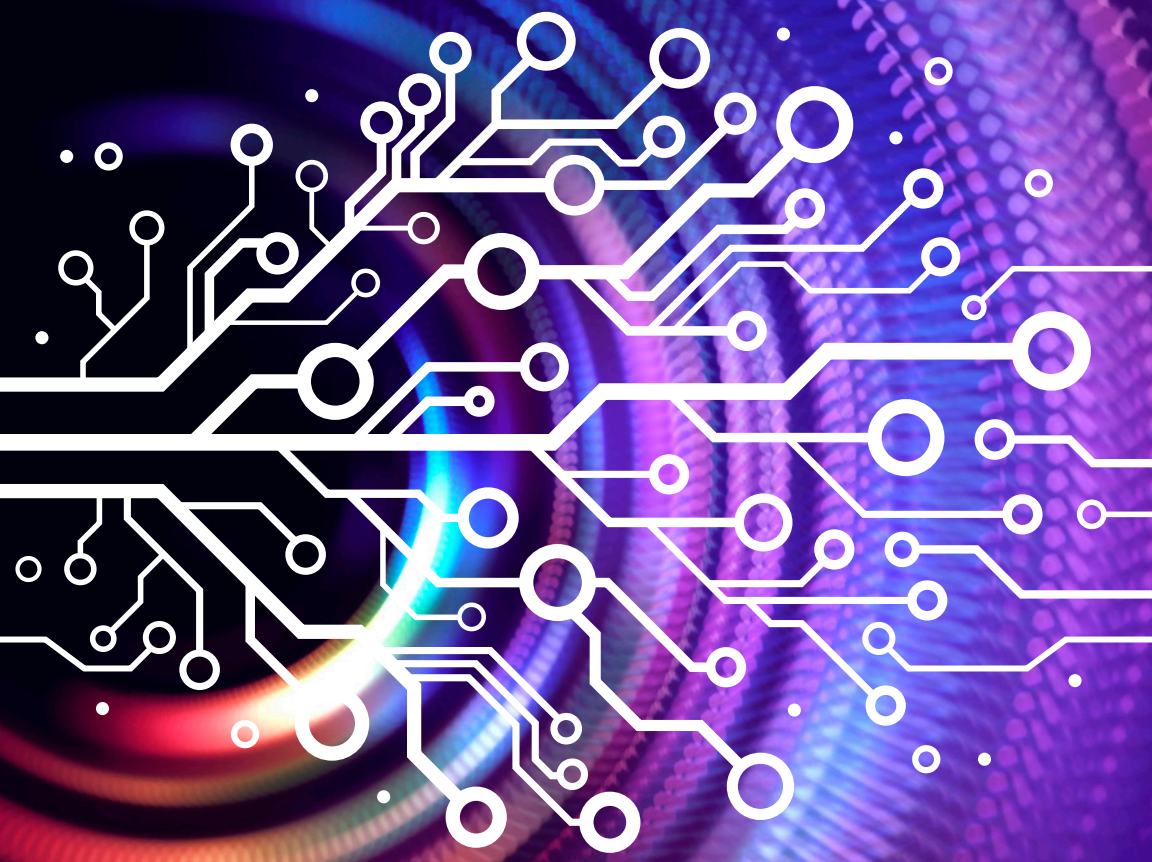
DELL Technologies

MODERN ENTERPRISE DATA PIPELINES

A Dell Technologies perspective on today's data landscape and the key ingredients for planning a modern, distributed data pipeline for your multicloud data-driven enterprise



Mike Bachman | Haji Aref | Rick Lemelin | Andrei Paduroiu

MODERN ENTERPRISE DATA PIPELINES

A Dell Technologies perspective on today's data landscape and the key ingredients for planning a modern, distributed data pipeline for your multicloud data-driven enterprise

Mike Bachman | Haji Aref | Rick Lemelin | Andrei Paduroiu

This paperback edition is published in 2021 by

Dell Inc.

1 Dell Way Round Rock TX, U.S.A.

www.Dell.com

Printed by:

Shutterfly Business Solutions
7195 S. Shutterfly Way | Tempe, AZ 85283 | (480) 494-4266

Printed in the U.S.A.

ISBN: 978-1-7373623-0-2

Contents

FOREWORD

By Michael J. Morton

In this modern era, we all live on Planet Data. It is more challenging than ever to come up with an instance in our daily lives for which our actions as humans are not ultimately producing data in some way. The very same can be said for a business. But now, for a business to be competitive, it needs to continuously evolve the use of technology and techniques to generate the highest value insights and outcomes from a vast landscape of datatypes and sources. This is why an understanding of modern- and enterprise-scale data pipelines is critical for the data leadership of a business today. This book provides the collective knowledge and industry expertise of the top data experts of Dell Technologies to educate and help guide your company on your data transformation journey.

Perfection of data processing architecture and solutions is not an achievable state because of the rate of change when it comes to data. Therefore, education from skilled practitioners is more important than ever to help you build a foundation that can evolve and react as fast as the pace of our data planet. This book covers a lot of ground and can serve as guide to help you become more familiar with the latest technology considerations, while also not ignoring the reality of the technologies that businesses are dependent on today.

As the Dell Technologies VP of Data Management and Intelligence, and a Dell Technologies Fellow, I am honored to have had the opportunity to endorse this book.

AUDIENCE

This book's intended audience is technical stakeholders and decision-makers for an organization's data architecture strategy, but its content is intended to be accessible to any technologist or business analyst. While we focus on enterprise solutions, the key concepts are applicable to businesses of any size.

AUTHORS

The authors of this book are engineers and architects from across Dell Technologies who share a commitment to help customers transform their organizations by bringing humans and technology into a productive balance.

Mike Bachman—or Bachman, as his colleagues call him—is the Chief Architect in the Office of the CTO (OCTO) at Boomi, a Dell Technologies business. Bachman consults with the Dell Technologies Enterprise Architecture Group, which is responsible for the company's largest global enterprises. He has over 20 years of professional experience in technology consulting, including infrastructure architecture, application integration, performance analysis, data strategy and emerging technology. In the OCTO, he investigates leading- and bleeding-edge technologies and applies them to a variety of real-world challenges.

Haji Aref has over twenty-nine years of experience with SAP development and advanced technical support for the following platforms: DB2, Oracle, SQL Server, Sybase and HANA. For the past fourteen years, Haji served as Chief Global Architect across Dell Technologies companies with a focus on Advanced Analytics, Virtualization, Data Strategy, big data, AI/ML, and IT Resource Management, Security, Specialized SAP Services to support Dell Technologies customers directly, as well as provide Dell Technologies SAP Research and Development and strategic account support. Haji holds a Bachelor of Science degree in Chemical Engineering from Binghamton University. He speaks 6 languages. Haji is very involved in medical research and programming nanobots for cancer research. Haji is a member of the SAP Developer Council. Haji is a distinguished enterprise architect who holds Dell Certified Enterprise Architect – Level 3.

Rick Lemelin, a certified Distinguished Enterprise Architect, has worked in Information Technology for 31 years, across all facets of business and technology optimization and transformation. He specializes in complex large-deal solution framing while delivering innovative approaches to integrated enterprise architectures. Currently, Rick leads the Dell Technology Select Architecture Strategy and the Enterprise Architecture Center of Excellence.

His team leverages multicloud architectures to enable clients', digital business-enabling, transform outcomes. Rick is driven to understand, and teach, the divergence between legacy and digital business models, including the requirements of next generation: application/data behavioral patterns; infrastructure topology deployment patterns; integration patterns; data sharing patterns; and resource consumption patterns.

Andrei Paduroiu is a Distinguished Member of Technical Staff within the Unstructured Data Storage group at Dell. Author of several US patents in the event streaming space, he is one of the founding members of open-source project Pravega, a revolutionary streaming storage system that aims to unify stream and batch processing. Andrei earned a Master's degree from Northeastern University, concentrating on machine learning and information retrieval, and a Bachelor's degree from Worcester Polytechnic Institute.

About Dell Technologies

Dell Technologies (NYSE:DELL) helps organizations and individuals build their digital future and transform how they work, live and play. The company provides customers with the industry's broadest and most innovative technology and services portfolio for the data era.

About This Book

We conceived the ideas for *Modern Enterprise Data Pipelines* almost two years ago, well ahead of the COVID-19 pandemic. Originally, we wrote and released this book as an internal white paper to help shore up any knowledge gap that Dell Technologies engineers and architects may have had. It was clear to us then that Dell Technologies practitioners, while highly knowledgeable about infrastructure concepts, needed to understand data better. At the top of the list were data fundamentals, how data flows within a process or application, and the technical factors that can increase the probability of the customer's successful digital transformations. This agenda was ambitious, so we created a technical, yet consumable, approach to help them begin to think through data architecture and pipeline engineering. Then COVID-19 happened.

While COVID-19 has had a disruptive effect on business practices, fomented decentralization of global supply chains, and impacted in-person human interaction, one thing is certain: COVID-19 did not stop the creation or consumption of data. In fact, data continues to grow at a staggering rate commensurate with digitization. According to McKinsey, digital adoption is not only still happening, it is happening at a pace that no one could have reasonably predicted before the pandemic[1]. This means, of course, that with rapid digitization comes data; a lot of it, and fast. In a way, the pandemic has spurred a palpable sense of urgency around understanding data. This is because the pandemic has fueled a vast increase of new nodes—disparate sources of data—and the edges that connect them. We believe that transformation is now a requirement for organizational survival in a post-COVID world.

The pandemic has changed the way that data is created and consumed; but the fundamental concepts remain the same as they were before the disease emerged. For this reason, we decided to expand our white paper into a book available to anyone who wants to acquire the core concepts necessary to understand data, data pipelines, and the data fabrics that these pipelines compose.

1 https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/the-covid-19-recovery-will-be-digital-a-plan-for-the-first-90-days

Our focus is data pipelines: what they are, why they are important, and the thinking needed to construct them. While we provide sample implementations of data pipelines including certain Dell Technologies products and software solutions, this book is not intended as a product guide. Instead, it is intended to give you an understanding of pipelines and the variety of data that flow through them. These concepts are applicable not only to enterprises but to businesses of any size.

We place data pipelines within a generalized framework for an enterprise data architecture that can support a wide range of applications with unique and future-proof efficiency and power.

Specifically, you will learn key concepts of a multicloud data fabric, different architectures for batch and real-time processing, input and output source and target data, and how these ideas come together to provide an understanding of end-to-end data pipeline construction. Once we've described the features of pipelines and data fabrics, we go on to explore how pipelines feed artificial intelligence (AI), and how AI and Machine Learning (ML) play an enormous role in the future of every organization. Finally, we show how the goals for business and technical stakeholders throughout an organization can take advantage of pipelines built with Dell Technologies solutions and expertise.

In *Modern Enterprise Data Pipelines*, we also seek to provide a common vocabulary so that all stakeholders within an organization can speak a common language. This way, technical alignment to desired goals can be achieved regardless of whether process stakeholders come from business operations or IT. We expect that with the concepts presented here, you will be in a good position to challenge convention and to ask deeper questions about how your organization can achieve its technical objectives, transforming and thriving in the new data age.

Finally, although the book is intended for technologists, we have kept the tone informal and conversational, with the hope that non-technical readers will find the concepts more approachable.

ACKNOWLEDGMENTS

Thank you to the following people who either supported this publication or dedicated hours or days to read our prose and validate our assertions:

Michael J. Morton, John Distasio, Dave Schroeder, Tom Roloff, Will Corkery, Mandy Dhaliwal, Rajesh Raheja, Ed Macosky, Flavio Junqueira, Thameem Khan, Shawn Raess, Premjit Mishra, Csaba Lorinczy, Jennifer Savoia, Mike Kiersey, Scott DesBles, Chris Larsen, Alejandro Flores, Frank Lentini, Mark Clifton, Harvey Melfi, Steve Tseng, Chris Timmerman, Fran Parras, Dirk Masuch, Bo Moller, Rene Klomp, Simon Hildesley, David Irecki, Russ Caldwell, Ted Schachter, Dave Hock, Ben Koehler, John Shirley, Srikanth Satya, Josh Raw, Ashish Batwara, Preston Peterson, Tim Plaud, Dave Lesshafft, Joe Flynn, John Reeves, Ravi Krishnan, Clayton Stoering, Kristina Avrionova, Carlos Castro, Adam Arrowsmith, Matt Krebsbach, Bernie Carson, Lesley Nispel, Andy Tillo, Chris Cappetta, Mrnalini Vasanthi, Chris Cassidy, Ranjana Haridas, Lee Sobotkin, Brad Detlefsen, Mike Frazier, Paul Nuschke, Alicia Malboeuf, Joel Alonzo, Myles Suer, Stephanie Abrams, Raju Jaini, DJ Krebsbach, Joe Vukson, Stephen Richards, Jeannine Drost, Chris May, Jessica Gomez, Debra Slapak, Angelia Mcfarland, Sean Evans and Andre Gibbs.

Your insights are valuable and continue to help shape our thinking.

Also thanks to the Dell Information Design and Development team under Jerele Neeld and Andre Pellet, who provided book editing, illustration, design and production: Steve Bracamontez, Ramji K, Doug Kaiser, Thomas Mathew, and Hilde Weisert.

CHAPTER 1

INTRODUCTION

This book provides a Dell Technologies perspective on data pipelines within a modern, multicloud data architecture. Data pipelines feed various applications. Applications act as a portal through which users—external or internal, human or automaton—interact with the processes, data objects or data models of the organizations that build them. It is through this interaction that data is curated, staged, transformed and re-presented as information. When this data moves through a resilient architecture it can be transformed into value for both data consumer and producer.

What is a *data pipeline*? **Data pipelines** are abstracted software representations of physical or logical compute and long- and short-term memory systems. Data pipelines can be represented entirely in code, or made up of connected composite endpoints, located in any cloud, that perform at least one function before raw source data becomes processed *target* data. While data pipelines access physical-layer infrastructure at some point, the type of architecture that is discussed in this book involves pipelines close to the application. In other words, data pipelines are transformational processes that involve sourcing data from one location, processing the data through some ordered sequence of actions, and targeting the refined data to a desired destination.

Data pipelines are composed of a set of connected endpoints—nodes connected to a data network. The aggregate collection of these endpoints, the functions they perform, and how and where they maintain state, comprise a *data fabric*.

A **data fabric** is an architectural concept for an organization's comprehensive set of endpoints, services and processes that can be used in any combination or permutation to achieve digital outcomes. A data fabric includes infrastructure layer features like memory systems, processors, and networks (for example, data streams, lakes, warehouses and marts). Finally, a data fabric, as implied by its name, contains all organizational data (metadata, logs, records, images, documentation, and so forth), architecture, and the resources to govern them. It is within the data fabric that pipelines operate.

If a data fabric is a tapestry, each data pipeline is a thread. Properties of these threads can include an image and an endpoint, service or function.

An organization that chooses to transform its business requires a data architecture supporting a wide variety of data sources, a robust processing core that can scale to any demand and a way to tie together data from different locations and contexts to achieve a desired outcome. Well-constructed data pipelines and the data fabrics in which they are created enable organizations to transform and thrive in a multicloud environment. With this approach, organizations can serve their consumers in an appropriate context, anywhere in the world.

After reading this book, you should take away the following:

- **The changes in thinking that are necessary for true transformation.** Transforming an enterprise's data pipelines begins with a change of mindset, culturally and architecturally. Unlike legacy 3-tier approaches to information systems of the past, the shift to a multicloud strategy requires thinking *small* about applications or services while thinking *big* about data. Chapter 2 focuses on these topics by introducing and discussing the organizational concepts of *Estate 1* and *Estate 2*.
- **The vocabulary, architectural foundations, and conceptual framework required for a technical transition.** With the aid of a logical, conceptual organizational framework, an understanding of modern technology, and pragmatic approaches to their use, distributed applications and data patterns are now prime candidates for a technical transition. Chapters 3 and 4 delve into architecture patterns that ingest raw data from anywhere, process data through sequenced steps, and stage processed data to the consumer. This becomes a patterned conceptual framework that we call *input–output pipeline architecture*. Applying this framework to your current estate can help you achieve your desired future estate.
- **An understanding of how to apply data pipelines to transformations**. Finally, we show how Dell Technologies solutions for multicloud data pipelines can help organizations achieve successful transformations, not only with the physical layer servers and storage for which Dell Technologies is known, but with the software that allows applications to be built, secured, and managed in any cloud environment. This includes ways to discover, harvest, and connect to source data from anywhere within the estate, to process data through persistent and intelligent pipelines, and to present the data to users, applications, and services through self-service API resources to accomplish the desired outcome.

In summary, our goal is to provide you with an understanding of data pipelines to guide your thinking about the design, implementation, and operation of a modern data architecture. Whether you are a technical or a non-technical reader, the vocabulary and framework presented here is intended to enable you to distill potentially confusing terms into understandable and usable principles that your organization can apply immediately to a meaningful digital transformation.

CHAPTER 2

A New Way of Thinking about Data Architecture for the Cloud

A deluge of new technology is in the market and competing for your attention. One company promises to "disrupt the market" with an esoteric product feature, while another proposes to "change the way business is done." If it seems like there are a greater number of vendors in a specific focus area than ever before, it's because there are. In a cursory Internet search of technology startups, it doesn't take long to find several fledgling companies crowding into burgeoning markets. As an example, Figure 1 shows the panoply of artificial intelligence companies in Canada in 2019.

This is a startling number of companies in just one country in an industry that is relatively new. What has made this kind of growth possible? To a great extent, the cloud.

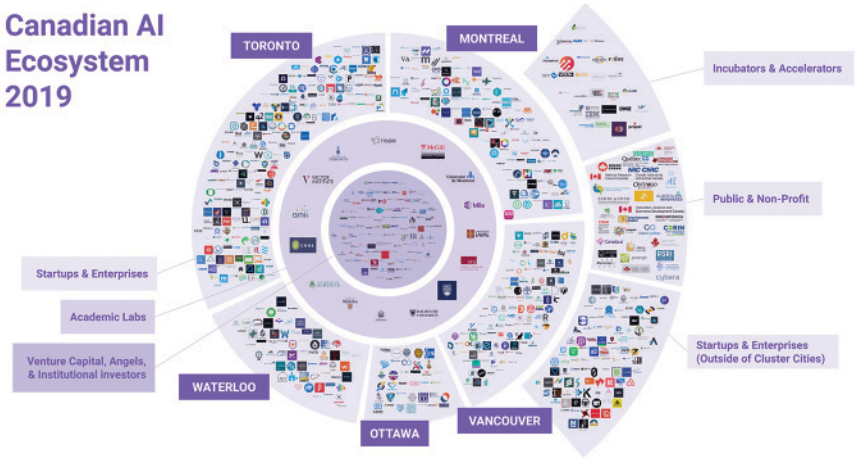


Figure 1. Canadian AI companies, startups, labs, incubators and investors in 2019[2]

2 Gagné, Jean-Francois https://jfgagne.ai/2019-canadian-ai-ecosystem/ medium.com/@jfgagne. 2019.

The cloud has enabled many businesses to emerge seemingly from vapor. What began as a great place to store songs and view videos has turned into a business incubator for small, independent startups. And each startup has all the tools necessary to build a successful business from the ground up—or rather, *cloud down*.

Building a business from the cloud gives entrepreneurs with good ideas and the ability to write code much of what they need in order to compete with large and established incumbents. They can do this, too, with almost none of the burden borne by an established enterprise. Free of what we will call *tech debt*—costly brick and mortar storefronts, and the need to hire a large staff—a startup can be efficient, lean and focused. Application development decisions can be made with the latest modern architecture and engineering design patterns.

In this environment, how can established enterprises transform and compete with startups? On one hand, established enterprises face impediments to transformation, such as tech debt. On the other hand, established enterprises can seem unbeatable. They have access to a great deal of capital and multitudes of customers. They have established processes and practices, market knowledge and experience, seasoning and scale. They also have a great deal of data. Can't large incumbents use this to their advantage against startup challengers?

Conversely, how can a startup increase the probability that it will survive? In the next 12–24 months, how should it continue to compete with established enterprises that are building, buying, or bolstering their own competitive countermeasures, while fending off increasingly younger companies within their ever-crowded market?

While a complete answer to these questions is beyond the scope of this book, we can provide a change in thinking for the established enterprise as well as the startup. This change begins with looking at data as a vital natural resource in a modern IT landscape. We refer to the legacy environment as *Estate 1* and the modern environment as *Estate 2*. The architectural pattern differences between these estates will provide the understanding of what it will take for your business, organization, or enterprise, whatever its size and history, to compete well within the multicloud data economy.

The Two IT Environments, Estate 1 and Estate 2

When we refer to ***Estate 1***, we mean a legacy platform of technologies that is used as a base upon which other applications, processes, or technologies are developed. A legacy platform consists of silo-based infrastructure domains supporting legacy, persistent application stacks. These environments may

include both physical and virtualized hardware; however, the underlying architecture still follows an amalgamation of decisions made over the past decades. Characteristics of Estate 1 include, but are not limited to, these elements:

- Workload pinned to physical blades
- Traditional HA/DR/Failover/Failback where resiliency is at a hardware level
- SAN with traditional storage arrays
- Multiple levels of Access Control List (ACL) groups or DMZs
- Physical load-balancers and firewalls
- Isolated network domains, and
- Fixed resource allocation

Thus, application programs written for one platform will not work on a different platform. Examples are persistent workload images or a CapEx run model. The existence of server, storage or network virtualization does not itself equal a software-defined data center (SDDC).

Estate 2 refers to a fully enabled, multicloud-ready, SDDC-supported, native cloud application-workload framework, like Kubernetes. Characteristics include, but are not limited to:

- Cross data center/cross cloud dynamic/flexible resource pools, workload domains, elastic scalability, and availability zones
- Network function virtualization (NFV)
- Micro-segmentation
- Geo-fencing
- Secure tenant isolation
- Resiliency via availability zones, not hardware-dependent
- Any-to-many dynamic API connectivity between any PaaS, SaaS, FaaS, BPaaS, and so forth
- Dynamic/variable resource allocation, sizing and utilization
- Non-persistent workload images, such as a serverless OpEx run model

Estate 2 is always a net-new, fully-enabled hardware/software architecture built to support net-new workloads or services. It is inclusive of proper CI/CD/DevOps-focused resource automation, orchestration, life cycle management, and brokerage.

Workloads and the framework of Estate 2 can run on different platforms within any infrastructure landscape, whether on-premises, cloud, or hybrid, or any

combination of these. By running within or across any of these environments, Estate 2 is inherently a multicloud architecture.

Before we discuss multicloud further, here is a short refresher on the meaning of *cloud.*

Cloud can be loosely interpreted to mean any data center environment with *abstracted* physical computational or processing, networking and storage resources that host applications, services and data. The abstraction of the physical resources, in the form of services, allows for a software-defined strategy to separate the application from the data. This means that not only do hyperscalers, like Google Cloud Platform (GCP), Amazon Web Services (AWS), or Azure get to call themselves "cloud providers," but, in our view, any enterprise with the ability to deliver services can do so as well. The implementation details may be different; this allows for a Service-Oriented Architecture (SOA).

Figure 2 shows an example of three types of Estate 2 architectures in which multicloud or any-any cloud infrastructures obtain applications. Data is separated from the application in each architecture. Services (or daemons) and functions are either tightly coupled to form the application (as in the SOA pattern), or loosely coupled in the micro- and nanoservices pattern.

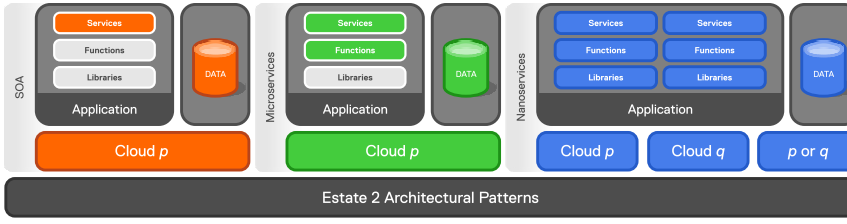


Figure 2. Example of three Estate 2 architectures

The SOA model, while available as Infrastructure as a Service (*IaaS)* in the hyper-scaler environment, is a common manifestation of an on-premises environment, as well as IaaS. Business logic (functions) are bound to services with tight coupling with the operating system, meaning that the application logic, libraries and services are associated with one operating system on a virtual (or bare metal) machine. Data is separated logically, via software-defined storage and networking.

A microservices pattern is like the SOA, in that the architecture abstracts physical infrastructure into the virtual. The differences come by way of breaking down the application layer into functions and services, so that the dependency is no longer based on the operating system to bind services directly to the application functions. The functional layer at a logical level (usually as a container or a VM) is then free to house the libraries and code

to instantiate and run the application, and host services can be called when needed. The microservices framework allows for more granular control of the logical infrastructure supporting the application and eliminates dependencies between the code and the infrastructure resources to run the application code.

Microservices enable the modularity required to swap in and out services components for a given build. This allows for scale, efficiency, and wide distribution of an application's infrastructure, from whichever cloud where it runs. Furthermore, a microservices architecture enables independent development of the components of the application, versioning, and proper isolation of the services that are designed to communicate with each other, while using logs to manifest state for any type of stateful services.

Taking a software-based approach to best-of-breed services, regardless of the cloud, is the central theme of *nanoservices*. *Nanoservices,* also known as Functions-as-a-Service (FaaS), Lambdas, Infrastructure as Code, or Serverless computing, are a highly optimized way to employ just-in-time endpoints to a pipeline.

The goals of nano-based architecture are ambitious. Not only does "nano" mean that services from any cloud can become swappable constituents of infrastructure, but this also means that the business logic can be interchangeable as well. Further, services and logic can be ephemeral. This translates into the application only consuming the resources of the compute, data, memory system and network resources when called.

A successful deployment of nanoservices means rethinking the architecture of the application itself. An example of this is a *multicloud composite application* (MCA). In an MCA, endpoints can be present in one or many different clouds. Each component of the MCA could be in a different cloud, attempting to make calls to dislocated data over an IP network. While an MCA composed of endpoints whose processing occurs in completely separate clouds isn't necessarily the best idea to use case—and probably something to avoid altogether—having an application comprising moments in time, independent of state, could lead to the ultimate best-of-breed app for the most cost-effective use of the resources necessary to run it. Want the best support vector machine library to train your data, today? Instantiate a function from Cloud p. If, tomorrow, Cloud q's logic improves enough to run in the application instead of its processor, then switch. Want them both? Use them both.

This type of thinking underscores the architectural philosophy of a modern enterprise. At Dell Technologies, we refer to this modern enterprise architecture as Estate 2.

Whereas Estate 1 is concerned with the type of physical infrastructure upon which applications store and process data, Estate 2 concentrates on *how* infrastructure can be abstracted so that applications can efficiently store and process data from any cloud to any other. This means that not only are the application and data decoupled from the physical or logical infrastructure,

but the application and data are decoupled from each other. Additionally, the notion of an application, itself, may be a departure from tradition. Application state is ephemeral, and its composition is a sequence of functional calls to services that access the data or its derivatives when necessary. In other words, an application can be a collection of calls between services between clouds.

If you're thinking that Service-Oriented Architecture, or its more modern cousin, microservices, already handles Estate 2 concepts, you are correct. But Estate 2 is more than SOA and microservices. Estate 2 is concerned with any composite set of services or functions that is authorized to access any other composition of services or functions. Since data services interact with the data, and because services can be subject to improvement from one cloud to the next, the data should be accessible to any cloud's service at any time. Importantly, this type of dynamic workflow needs to be free of disruption. This means that automation must be in place to ensure that the elements of Estate 2 architecture—functions, services and data interactions—occur as quickly as possible, even when components change. John Roese, CTO at Dell Technologies, alludes to this kind of model for emerging tech, such as autonomous vehicles.

"What's interesting is that the system is not just going to be idle; data is going to flow across the entire system in very dynamic ways…"[3]

— John Roese, Dell Technologies Chief Technology Officer

When considering a model for a modern data architecture, SOA and microservices tend to focus more on the decoupling of data services from the application than on the composition of decoupled applications to dislocated data. This may seem like a trivial nuance, but in a modern landscape where cloud platforms are competing to keep customers from defecting to other cloud providers, it is critical to success. The modern application is no longer a simple stack of 3-tier architecture—*model-view-controller (MVC)*[4]—within a neat package. Instead, the architecture may be the entire domain of one service but completely missing from another.

Key Concepts in Transformative Thinking

Digital transformation can be a difficult journey that is often fraught with barriers to success. A 2019 Forbes article alludes to a high failure rate of enterprise digital transformation stemming from long paths of attempting to transform

3 https://www.zdnet.com/article/autonomous-vehicle-delivery-impossible-without-extreme-automation-dell-cto/

4 https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

legacy processes into something more modern.[5] Years-long projects lead to "digital transformation fatigue," and companies desperate for a solution may focus too heavily on trying to innovate like disruptors. Unfortunately for those established organizations, their business model may not be set up to do so.

In what ways, then, are established enterprises trying to keep up with new companies, and why are their transformation initiatives failing? In some cases, the established business models of an incumbent enterprise may not have the financial maneuverability of a startup.[6] In other cases, organizational leadership fails to clearly articulate transformational goals and empower their employees or provide them with the right skills to enable transformations.[7] One *technical* impediment to transformation is *tech debt*. While not the only reason that digital transformation efforts flop, tech debt is an enormous hurdle for legacy organizations to overcome.

In the next section we will look at the importance of tech debt and an organization's current dependencies on existing systems and business- and mission-critical processes that prevent them from succeeding in transformation. A common cause is the enterprise trying simply to offload their large amounts of technical debt to the cloud. In other words, when organizations try to migrate an Estate 1 ecosystem into an Estate 2 landscape, failure is likely.

Tech Debt as an Obstacle to Transformation

Tech debt is a metaphor used to describe the amount of time, effort or cost it takes to rework a previous, "shortcut" solution.[8] While the term is typically used for software refactoring, it can also be applied to summarily moving IT operations to the cloud.

For established companies with large amounts of technical debt, transformation can be especially challenging. The odds are stacked against an organization that expects to port its existing architecture to an "all-cloud" infrastructure, a burdensome undertaking. Consider environments that can possess a great deal of "big iron" that supports legacy applications running on heritage code, and whose supporting processes are under- or un-documented. The existing problems that may be embedded in Estate 1 cannot simply be remedied by moving it to an Estate 2 paradigm in the cloud. As our customers on the transformation journey know all too well, under- and un-documented, data-siloed, bespoke services and monolithic applications of Estate 1 cannot practically be moved to the cloud because of a corporate mandate. Many of the anecdotes that describe wholesale moves to the cloud center on an

5 https://www.forbes.com/sites/forbestechcouncil/2019/11/01/why-your-enterprise-keeps-failing-at-digital-transformation/#37ad0eb7258f

6 IDG 2018 State of Business Transformation - https://cdn2.hubspot.net/hubfs/1624046/Digital%20Business%20Executive%20Summary_FINAL.pdf

7 ibid

8 https://en.wikipedia.org/wiki/Technical_debt

Estate 1 move to an Estate 2 environment.[9] Experience shows that expecting to convert mainframe applications to the cloud and simply map legacy database schemas to SaaS objects rarely if ever works.

Why is this? Many applications, services and data sources have not been properly re-platformed for cloud environments. Many if not most older applications, such as mainframe applications, were built without such notions as functional application programming interfaces (APIs), abstracted hardware, dislocated APIs and libraries, disjointed networks, shared databases, object databases, ever-expanding metadata, or verbose log streams.

Even the Internet itself has become something entirely different today than when it was first conceived. In the 1990s, an individual's Internet experience was ephemeral. One would "hop on and off" of it much like riding the Tube in London. Content was scarce then and so, too, were the ways to find it. Data was created and the amount grew, but companies could readily manage it, usually within a single database or a database cluster. Few web-based applications existed, and those that did had limited function.[10] Early adopters of the Internet began writing applications and protocols to make the web a much more "inhabitable" place to visit. As more Internet users visited, more data was created, and more sophisticated applications proliferated. New apps meant more data. More data meant new apps.

Throughout the 1990s and into the early 2000s, a new generation of producers and consumers of content and applications cropped up literally everywhere. Companies were keen to capitalize on this vast market. Data began growing at exponential rates and, thus, was no longer as easy to wrangle. Instead of managing one database or even a small cluster of them, data storage needed to be rethought altogether. The Internet was no longer a place to *go*; it was now—and still is—an alluring place to *be.* And because of this the need to provide global scale compute and storage solutions was required. Enter *the Cloud*.

Today, we are constantly online, connected by new tools (such as smartphones, IoT (Internet of Things) devices, and so forth) to a web of rich kinds and massive amounts of data. And it is the data that enterprises are after. In the early days of what we now know as *the Cloud,* hyperscalers we know of today were placing bets on the future of global reach. Stemming from their early days as book sellers, index engines and builders of operating systems and productivity applications, and yes, even a young college student who was building computers out of his dorm room, perhaps it was difficult for many to see then the wave of resources that would be needed to harness data. As a result, whole new industries were built from massively scalable data centers, using armies of developers to create more and more applications to spawn vast new wells of data.

9 https://www.forbes.com/sites/jasonbloomberg/2018/08/05/dont-let-your-cloud-migration-become-a-clusterfck/#2bb738113632

10 https://www.webdesignmuseum.org/web-design-history

Initially, it may seem that the clear winner is the enterprise with the most resources to process and harness data while producing new ways to harvest it. If cloud hyperscalers have the most "toys," then logic may dictate that an organization should shift its assets—applications and data—to the cloud. For some, transformation to the cloud may work; but for the vast majority, it does not.[11]

Harvesting, processing and harnessing data growth at scale becomes challenging if not downright untenable. To underscore this, moving wholly to the cloud is not necessarily for every enterprise; it's certainly not for every application. And one important reason for this is that these applications come principally from the Estate 1 camp, with all the tech debt that comes with it.

An alternative to transforming is *not* to transform. This, too, is ill-advised if your company wants customers. After all, "customers never tell us that they want *less data slower*."[12] To keep up with the speed, scale, and scope of data in a modern world, simply doing business as it was conducted over the last generation or two is also not the answer. The rapid rate of application usage and data creation and growth has driven new ways to conduct commerce and produced entirely new industries. If last century's enterprises cannot adapt to a new model required to meet the demands of the employees, customers, and partners of tomorrow, then their future is bleak. Look at the relatively recent tombstones of what, in the 20th century, was the engine of American innovation and business, such as Kodak™ and Xerox™.

Transformation is both meaningful and achievable if the whole company stands by it. By transformation, we mean the continuous process by which an organization can adapt to radical shifts in technology, relative to serving its community of users without interruption, while interacting continuously with trading partners and attracting and retaining the best possible employees. Transformation is a tall order; but it is necessary to thrive and even survive in business.

Why Is Transformation So Important, Now?

The cloud has enabled data and applications to start anywhere and scale everywhere. Creating, pooling and analyzing data, then having downstream consumers interacting with it, can be daunting.

We tend to be online constantly on an always-on Internet, with endless ways to connect to it. Smart objects and applications with little to no perceivable downtime allow users to interact with their families, friends, colleagues, strangers, apps and bots. There is widespread adoption of the cloud and

11 https://www.forbes.com/sites/cognitiveworld/2019/01/07/why-most-digital-transformations-will-fail/#37980516520e

12 Steve Wood, former Chief Product Officer, Boomi, a Dell Company: an often-quoted statement that we've learned to love and live by.

a smorgasbord of services offered within SaaS platforms (think Facebook, Spotify, Salesforce, AirBnB) or cloud hyperscalers' portfolios (for example, Amazon, Google, Baidu, and Azure). Our time, minds, and wallets are inundated with ways to interact with applications found within this ethereal landscape. And the *data exhaust* that we, as consumers, leave behind with every click on a link, pause on a webpage, or purchase from an eCommerce platform, become not only a traceable way to monitor what we view, listen to, watch, or buy; but a way for these enterprises to watch our digital personas interacting within a virtual world.[13]

In one sense, the enterprise transformations we are considering have less to do with operating a business within a new network, and more to do about creating a virtual instance of the enterprise in a completely different dimension. And this dimension is always on, receiving and processing data to make sense of the world with which it is interacting. It is an interface to our digital selves, and it is unprecedented. *This* is what "transformation" truly means: *How well can an organization represent itself faithfully in a way that digital humans and non-humans can do business with, be employed by, and interact with it?* All the while such organizations continue to provide our corporeal selves with brick-and-mortar storefronts so that we can remember what it is like to engage in the human-to-human commerce. In a sense, Estate 1 represents the last state of a known physical interface, where one could *know* the servers running the code to execute a given process, accessing data from a location to which a person could literally point. Estate 2 is far more abstract. And without the tools to bridge the two estates, failure is likely.

Outcomes First, Process Next

Another overused term is *outcome*. *Accelerating* outcomes. *Achieving* outcomes. *Business* outcomes. *Transformational* outcomes. Unfortunately, there is so much packed into the meaning of the word that, should we choose to overlook it, the likelihood of repeatable success plummets. Outcomes are critical to a successful transformation. In fact, the best way to begin a transformational process is at the end. Identifying the desired outcome *is* the beginning of the transformational process.

Outcomes typically take shape in several forms:

- **Increase** something (mind, market, margin, spend, investment, adoption, security, brand-awareness, NPS score, value, etc.)
- **Decrease** something (attrition, waste, time-to-value, cost, etc.)
- **Avoid** something (fines, fees, sanctions, penalties, loss, conflict, jail, suffering, etc.)

13 https://www.datasciencecentral.com/profiles/blogs/what-is-data-exhaust-and-what-can-you-do-with-it

Simply look at the patterns of any perceived outcome and ask *why*. Eventually, you'll arrive at one of the above outcomes. Continue to ask why and you may get to the *real answer*, as shown in the example below:

I want to transform my business.

Why? Because my business is losing money.

Why? Because it takes too long to resolve customer issues.

Why? Because my customers are on hold for an average of 17 minutes before talking to a human.

Why? Because we don't have enough personnel.

Why? Because there aren't enough people to hire in my area.

Why? Because my policy says that workers must come into a physical building. How would a policy adjustment affect the hiring of new employees? It might *increase* hires.

This example trivializes a transformation, and it may not solve the actual problem. The key point is the importance of an outcome.

Outcomes should be the starting point of designing a process. If the intent of the workflow is well understood, then it is possible to represent in software a model of the real world as a digital version—a *digital twin*. An accurate model simplifies writing an application, process, service or function. When the digital model mimics the real world, then flows of data from various sources can converge in such a way that an outcome can be *orchestrated.*

In the upcoming chapters, we will discuss what it means to orchestrate an outcome. Think of *outcome orchestration* as coalescing any number of pipelines whose data are processed—we call these *output pipelines*—in a way to achieve a desired outcome. Provided services can be called, connection interfaces are well-defined, data formats between endpoints are intelligible, and the data model is clear. Pipelines can be built to mimic the real world environment to achieve the desired outcome.

Understanding the outcome is essential to transformation. Without it, the objective is unclear. There is no way to compare the differences between the current and aspirational states. With clearly defined outcomes, however, transformations are achievable. The as-is and to-be pipeline differences are comparable. Attention can be focused on the appropriate endpoints to change, and clear and actionable build processes for all or some of each pipeline are possible.

In the next section we will focus on two primary architectural representations of data pipelines, Lambda and Kappa architecture. Dell Technologies deals with both; but we hope that, as we shift you from the *mindset* of transformations to the tools at the root of these transformations, we provide you with the vocabulary and understanding necessary to decide whether the tools that you have, or are open to using, will be sufficient to achieve your organizational objectives and desired outcomes.

CHAPTER 3

Optimized Data Architecture for Scale

In this chapter, we discuss two architectural patterns that are important to Estate 2, Lambda and Kappa architectures. If you are a Lambda and Kappa expert already, the fundamentals described will be a review. If not, they will provide you with a solid foundation.

We will raise familiar data topics and terms for reference before introducing you to the final area of *input and output* data pipelines in Chapter 4.

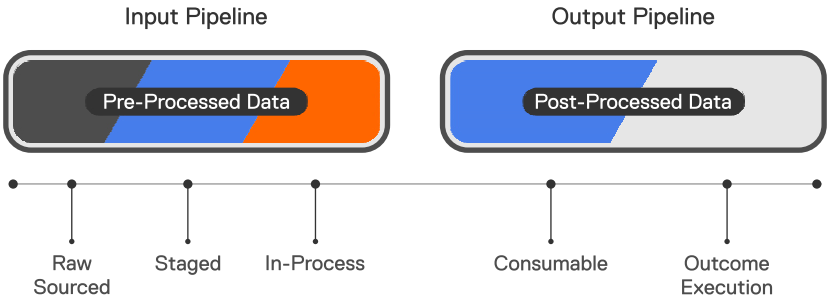


Figure 3. Input and Output Pipelines

Input and output pipelines provide a conceptual abstraction so that confusion can be minimized or mitigated as newer and more esoteric implementations of technology emerge. We use input and output pipelines to make data flows simpler to understand for information technologists, architects, engineers and business leaders, analysts and strategists.

A Brief Primer on Big Data

Data fuels the modern and future economy, and the information systems that run on this fuel are expanding dramatically. Humans and machines are generating, capturing, storing and wrangling data from disparate places and distributed environments in ever greater quantities. We are using data to classify the unknown, predict the future, and prescribe actions.

Those who can harness data at speed and scale will win markets, minds, and more. Organizations that can understand how transformations in a multicloud world will serve their customers, adopt transformational thinking and implement a multicloud approach to moving, managing, and monetizing the data that they cultivate will likely dominate data domains for decades to come.

By now, you have probably heard of the term *big data*. Perhaps big data conjures images of *Data* Lakes or, as we defined in Chapter 1, *Data Fabrics*—an evolved definition for a Data Lake and the multicloud ecosystem of possible endpoints that are able to transform data into context that is relevant for data consumers. Big data is industry jargon for large *volumes* of a *variety* of datatypes. And these datatypes are updated frequently; so much so that the *velocity* of change of this data may cause its veracity to be questionable. We just presented you with 4 *"v-words"—volume, variety, velocity, and veracity*—that embody the essence of the big data craze. Why is this important?

Data—big or otherwise—can (and should) be processed so that it can be analyzed, contextualized and turned into information that is then able to achieve a desired outcome.

In the past several decades, industries were developed to try to make data meaningful. From Online Analytical Processing (OLAP) environments to data warehouses and marts, massively parallel processing (MPP) systems, Hadoop and, now, inline streaming; different types of architectures that can make sense out of large sets of data may mean the difference between the success or failure of a company. Indeed, the speed and accuracy with which ever-larger sets of data can be analyzed creates competitive advantages for companies that decide to invest in the data processing platforms that enable new applications to respond instantaneously to their users.

As data continues its growth path to reach more than 175 ZB by 2025[14], organizations will face these challenges:

- Where they can access the data
- What sense they can make of the data
- How they can do something useful with the data

14 https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/#6d16bad35459

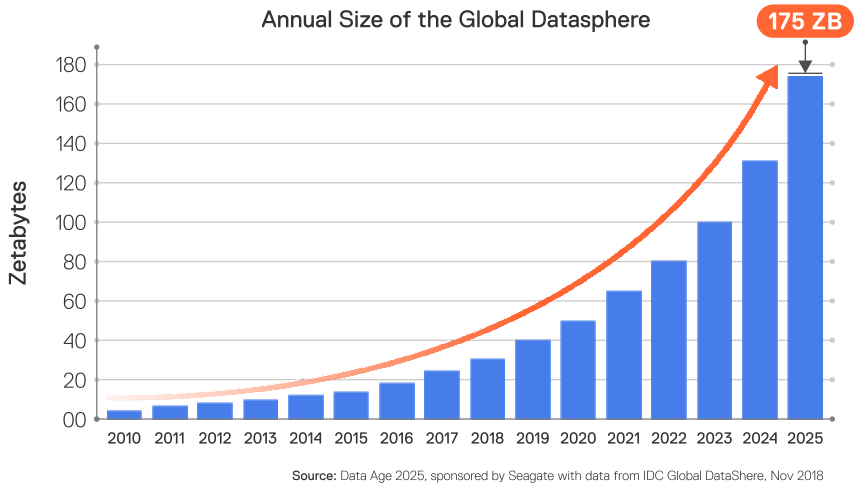


Figure 4. Quadratic growth projections for data through 2025

Our discussion of input and output pipelines will address this point in far greater detail, but before we get there let's understand Lambda and Kappa architecture, why they are important, and how they allow or prevent the optimal operation of data production pipelines in a multicloud fabric.

Lambda and Kappa architectures are patterns that describe philosophies of storing and processing data. Within each of these schools of thought are technical, political and cultural factors to consider, especially when considering the area of big data, analysis and data science. While this book focuses primarily on the technical characteristics of the different types of data architectures, the political and cultural factors are powerful, and it is important to understand them. First, let's get a working definition of Lambda and Kappa architectures.

What Is Lambda Architecture and Why Should I Care?

Lambda architecture involves two different types of data systems: immediate processing for transactions, followed by some form of analytical processing later. Consider batch processing and extract, transform and load (ETL) functions. Architectural models for data systems have been evolving slowly over time from mainframe applications and their local data structures, to client/servers with clustered data stores and centralized (and replicated) storage, to shared databases, where composite logical data sources are made up of physical pieces of the data (those fragments can be anywhere). Data distribution has come a long way. Unfortunately, the models with even the most geo-dispersed data are confounded by conventional architectures

that use methodologies developed decades ago, to solve problems that could not have foreseen the practical data challenges that exist today.

Lambda architecture describes a pattern of (big) data architecture that is scalable and fault tolerant.[15] By scalable, the original idea of Lambda architecture was to incorporate a scale-out (not up) approach to data services, as well as being fault tolerant for a wealth of workloads. Essentially, a Lambda architecture incorporates an element of data-at-rest and data-in-flight that is capable of growing and resilient in the face of failure, all without users of data knowing what's happening when they transact with the underlying data.

Essentially, a Lambda architecture comes in three layers: batch, speed (stream), and serving. New, incoming data is written to the batch and speed layers at the same time. This way the batch layer—a layer of data where there is a start and end to the data—can be processed at some point later, and stream processing can occur as close to real time as possible. The serving layer denotes the *processed* data that is ready to be presented to the awaiting applications.

Figure 5 illustrates Lambda architecture, involving multiple inputs and outputs for aggregation and processing, a batching layer and a real-time (speed) layer.[16]

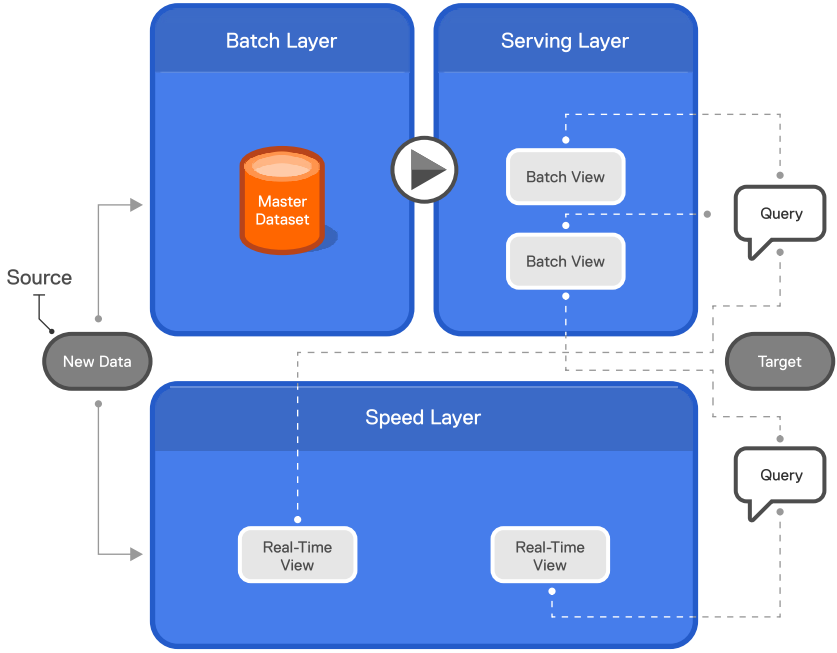


Figure 5. Lambda architecture

15 Data Lakes for Enterprises (location 1452)

16 From http://lambda-architecture.net/

You can think of the batching architecture like an archive, or a lower tier of storage where higher latency is tolerated, and the speed or stream architecture as a higher tier (where lower latency is mandated). Because of a batch's start and endpoints, it can be copied and loaded in full. For many use cases, a batch process is a large set of bounded changes from one time to another. We also refer to batches as "data-at-rest".

In a Lambda system, a *copy* of data coming is written to the "Speed" and "Batch" layers at the same time. This does not necessarily mean that a transaction from the source application's production database is written to another *analytical* archive data store at the same time. At some point *after* the write to the original data structure, replicas of the source data will be written to the Lambda system. This kind of atomic transaction followed by some analytical capability is commonplace today. After all, it is a best practice to not make copies of atomic data of a source transaction *before* it has been committed. It is certainly not a good idea to analyze data *before* an atomic transaction is written. Only in the immediate moments *after* the committed transaction is recorded should the data be eligible for analysis.

Data eligible for analysis is then eventually *batched* to one storage target or *streamed* to another as quickly as possible. Real-time analysis of the speed layer means that a copy of the atomic transaction manifests as a *state change*—that is, a change in the value of an element of data—known as an *event*. Events may be recorded to a variety of places; but one of the main areas to which they are recorded is an application log. Logs—whether application, system, or device—are what get processed typically through the "Speed" layer, so that a perceptibly immediate and temporally local analysis of data-in-flight can be conducted. This real-time layer is contrasted with the big data batches that are held until some later point in time—often at a time with limited impact to business—and then uploaded into a large archival data repository.

These copies of the data usually target two entirely different types of data-at-rest systems with different languages, semantics, schemas, APIs, file types and more. This is where concepts such as *Event-Driven Architecture (EDA), data streaming, Data Lake, Data Warehouse, Massively Parallel Processing, OLAP* and others come in to play.

Each of these types of data targets is a conceptual representation of different types of storage. For each system the storage is *structured, unstructured, or semi-structured*. Within each of these systems the encoding is suited for in-memory processing, others are Input/Output (I/O)-based, yet in others file systems or object stores are more appropriate. Additionally, the copies of the data source must be relevant to the data targets to which they are copied.

You may be thinking: “Who cares about this? We have plenty of data platforms and we make copies for reporting and analytics, already. That’s good enough.” These extra copies of data may not seem like a lot to manage. But it is one of the largest sources of problems for enterprises at scale. Why? Because the copies, themselves, are of the least concern.

Scale and change are two of the greatest contributors to problems of copies, and it exacerbates problems associated with tech debt. As data grows, copying, encoding changes, modifying schemas, capturing new data, new data connections, additional presentation-layer requests, data corruption, and a panoply of other items require more resources to manage them. Instead of solving a linear growth problem—O(n)—you have a quadratic problem—$O(n^2)$—or worse, an exponential problem—$O(2^n)$. Thus, scale is one of the greatest strains on the system and all the system and development time it takes to handle it. In practical terms, scale incorporates all the additional data sources and targets to manage them, developer coding time to connect the endpoints, additional durability and system redundancy required, and consistency checks that must be considered (and the time it takes to verify the consistency).

Kappa: Batches Need Not Apply

Kappa architecture, on the other hand, views all data as *data-in-flight*, which means that all data can be processed, all the time, in real time, without end—theoretically, of course. With Kappa, batching is not necessary. Thus, events and streams can be viewed as real-time processes where massive data moves are not necessary. How is this possible? Memory systems and computing technology are now sophisticated enough be used economically at scale, without sacrificing performance. While this book does not focus on hardware, we must point out that the physical layer technology enables software efficiencies of cloud-scale software. This means that non-hyperscale adopters can leverage virtual pipelines, integration and processing resources akin to cloud hyperscalers. And when an enterprise has comparable pipelines to GCP, AWS, Azure, and Baidu, barriers to a multicloud environment diminish.

In other words, imagine sourcing and processing data wherever you like, and presenting resulting data to any application built atop any service of any cloud. No need to process.

Figure 6 illustrates a sample Kappa architecture.

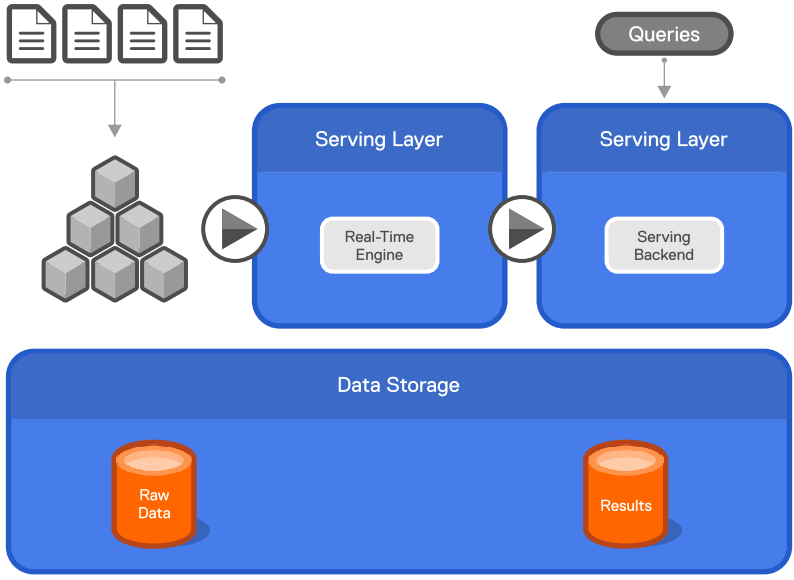


Figure 6. Kappa architecture example[17]

Data-in-flight is a more modern approach to data architecture, whereby all data is processed in real time. This means that all data is accessible, such that data from any data silo can be queried quickly and easily. Often people may refer to data/log streaming, event-driven architecture (EDA), and publish and subscribe (pub-sub)[18] as Kappa architecture; but these *are implementation patterns* for the Kappa architecture and not the architecture itself. However, pub-sub implementations are a widely adopted way in which distributed applications that have their data hosted within a multi- or any-cloud logical infrastructure is stored, for either Lambda or Kappa architectures. Recall that event-driven architecture (EDA, with pub-sub) is used in the "Speed" layer of Lambda architecture.

Data-at-rest means data that will eventually be processed. Historically, this kind of data has been within the realm of batching. By batching, this could be in the form of an analytical layer: business intelligence (BI), data warehouse, data mart, or data lake.

Kappa architecture avoids batch processing. What would it mean to your organization if you could eliminate the need to bulk load hundreds of GBs, dozens of TBs or more, at some interval of time? What if you could reduce the

17 https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry

18 Pub-Sub is a common implementation for EDA

need to *store* all that additional data, as well? In fact, organizations are closer than ever to realizing this approach, for a variety of use cases.

This is because of efficiencies provided by several layers of architecture:

- The scalability of a durable buffer
- The serialization of data within a durable buffer
- The presence of a high-throughput data-at-rest storage
- A common language for block, file, object and stream data
- Tiering of active and archive data as a first-class citizen of the Kappa Application Programming Interface (API)

What do we mean by *streaming?* Streaming is the idea that data can be processed in real time. There is a start to a stream; but no end. A data stream is an append-only byte-for-byte representation of changes in state of a system or process. As new data comes into a system, it can be collected, compared, de-duplicated, mapped, serialized, and made available to a calling application; all within seconds or less—often milliseconds. Real-time streaming is associated typically with engines that can persist data in some sort of persistent storage or *durable buffer*, such that multiple smaller events can be ingested with low latency and made immediately available for processing, and later be tiered into a data-at-rest storage system.

Traditional storage systems are optimized for high throughput, which they generally achieve by means of handling large writes well. However, trying to make many small writes to such systems does not bode well. There are not only throughput issues, but latencies large enough to detract anyone from using them that way. As a workaround, client applications have employed various *batching techniques* to be able to achieve high throughput with good latencies. In short, when handling very small writes (such as, 100 bytes or less), writers do not send those writes to the server right away; they fill an in-memory buffer of predetermined size or wait some amount of time before sending the entire contents of that buffer to the server as a single write.

Batching works well for systems that do not need to react to events in real time. The biggest downside to writer-side batching is that the system (readers) cannot act on events while they are in the writer's buffers. A solution for this is to create an *intermediate broker* (between the writer, reader and storage system), and perform in-memory batching there. The writer sends the writes immediately, regardless of size, and the broker would perform any batching before writing them to long-term storage. The readers would be served data from this buffer, which would solve the problem of reacting to such events in real time.

But wait! If the broker buffers in memory, there's always a chance it could fail before writing the data to its storage system. This means it could either lose the data (if it acknowledged to the writer) or cause data to be duplicated (if the reader processed it, but the writer re-sends it). These situations are known

as *at least once delivery* (duplication can occur) or at *most once delivery* (no duplication, but loss can occur).

It is for this reason that a *durable buffer* is required in the broker. As events come in from the writer, they are immediately written to a durable (replicated) storage, acknowledged to the writer and then made available for reading. Making the buffer durable (along with a few other techniques[19]) helps solve all these issues and can even pave the way for an *exactly-once streaming system*,[20] in which events are guaranteed to be delivered once, with no loss or duplication.

Hold on! The reason we decided to buffer in the first place was because the storage system could not handle small writes. Aren't we trading one problem for another? The answer is: *not necessarily*. While Lambda-architecture storage systems may be optimized for throughput, they are not the only type of storage system available. Log-based storage systems such as Apache BookKeeper[21] are designed specifically to enable *appending writes*, big or small, using extremely low latencies; with the caveat that the only way to read them is by replaying the log. Replaying the log can happen from any point within the log; but reading sequentially will lead to better performance. It is possible to implement a streaming broker[22] using a log-based durable buffer that offers low-latency ingestion for events of any size[23]. The events can then be replayed, in order, to be applied to any *data-at-rest* storage system in the background. From there, they can be accessed for historical processing or any other use cases that do not require real-time processing.

Data scientists need to connect to data sources easily, something that application integration systems could help accomplish, assuming the dataset is manageable; but the scientists will not likely be doing the connecting themselves. Instead, *data engineers* will be setting up the pipes. Since most of the data that the scientists need will be at rest in vast pools of storage, there is a need for data engineers to connect large pipes for batching terabytes of data at a time. Think of Hadoop—a file-based repository of data; data warehouses—structured repositories of data; or Massively Parallel Processing (MPP) arrays—strictly structured and refined schema processing. Each of these is an example of what many refer to as, in whole or in part, a *Data Lake*. Traditionally, applications and their transactional databases copy delta-changes (or all) data to these types of repositories on some periodic basis (often nightly). Once all the recently copied data is in the Data Lake, then it can be batch-processed with all the other data.

19 http://blog.pravega.io/2019/11/21/segment-attributes/

20 http://blog.pravega.io/2019/06/10/exactly-once-processing-using-apache-flink-and-pravega-connector/

21 https://bookkeeper.apache.org/

22 http://blog.pravega.io/2017/12/14/i-have-a-stream/

23 http://blog.pravega.io/2019/04/22/events-big-or-small-bring-them-on/

In a more recent shift in pipelining, data *in-flight* processing is a growing alternative. Another term for this is real-time streaming, or *event-driven architecture* (EDA), where data streams using log-based ingestion process data in real time. Queries can be conducted on the data immediately and continuously. This has numerous advantages beyond the scope of this book for scientists and consumers of data alike. Rather than batching terabytes of data; data can be "replayed" into new channels for deeper analysis that can be immediately actionable to upstream applications.

For example, a simple data processing pipeline made of four streams could start with an *ingestion* stream to which external events are appended. A *pre-processor* tails the ingestion stream, performs cleaning and deduplication, and writes the filtered events to an *analysis* stream. One or more analysis processors tail the analysis stream, detect patterns and produce output into the *action* stream (as events), which is finally tailed by whatever devices need to execute necessary actions.

Data: The Basics

Data. The word seems simple enough until you try to define it. Here is a dictionary definition:

> *"Information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer…"*[24]

Jeffrey Pomerantz, information scientist and author of *Metadata,* describes data (and metadata) this way:

> "Data is stuff. It is raw, unprocessed, possibly even untouched by human hands, unviewed by human eyes, and un-thought-about by human minds."[25]

Each of these definitions is far from complete; but at least they provide us with a starting point. We like to think of data as a representation of the physical world. Data can be descriptive. It tries to represent the world objectively in such a way that meaning can be derived from it. Data can have attributes, and those attributes may be subjective. It can be formless, or it can be bound to rules. Data manifests in a variety of ways: images, text, sound, symbols, and more. Data becomes valuable when turned into information. For the purpose of this book, we will focus on three types of data that we can turn into information:

- **Structured**: Schema on write. Relational database, tabular datastores
- **Unstructured**: No schema. Documents, audio files, video
- **Semi-structured:** Schema on read. Key-value pairs, XML, JSON

24 https://dictionary.cambridge.org/us/dictionary/english/data

25 Pomerantz (p.20).

Structured Data

A simple way to think about *structured* data is to think of the type of data that would be found in a table of a relational database. Each table contains a structure where a collection of column headers—a *schema*—defines the categories into which data are organized. With a schema, data are bound by rules that enforce how and where data are written, retrieved, updated, joined and deleted.

Access to data is governed through queries that are provided before data is accessed. This structure is common in many relational databases. Relational databases are often used in transactional operations for applications, with *Structured Query Language* (SQL) as the language most commonly used to access the data. Because data is governed by the structure of the schema that encases it, we will call this kind of data *schema-on-write*. This means that every record, transaction or event, must be organized into defined categories *before* a record can be written to the database.

Unstructured Data

Unstructured data has no schema. Examples of unstructured data are images, audio clips, video files, vector graphics, and Adobe Acrobat PDFs. Unstructured data is the kind of data with which humans interface and interact daily. Essentially, these files are simply accessed via a file system of some kind.

Semi-Structured Data

Semi-structured data has characteristics of governance offered by a database yet lacks the rigidity of a schema. In other words, values are stored, but instead of column headers and strict formatting, data are collocated with tags or *keys*. These keys provide a schema for the data; but there is no uniformity to the actual schema itself. This means that every record, transaction, or event may be unique. Only when a schema is read can one observe the differences between the types of data housed within them; and these data are often referred to as *self-describing*. For this reason, we refer to semi-structured data as *schema-on-read*. Examples of *self-describing*, schema-on-read data are JSON and XML, also known as *NoSQL.*

Each one of these datatypes is important to understand because applications rely on their structured-ness (or lack of it) to deliver value to the requestor. And depending on how data are accessed, each type of data has advantages. As described above, unstructured data works with everyday people better than other types of data, but automation performs better with structure. This is especially true for many actions including sorting, parsing, mapping, merging, joining, removing, and updating masses of data at a given time.

Regardless of the process, the *pipeline* into which any of these data are fed requires that the data get turned into information, and an early step of this transformation involves the use of a data model.

Important Data Engines

Data Models

Transactional, analytical, and master databases offer the framework for how applications interact with and store different datatypes. In simple terms, transactional data models deal with process for the real-time creation, reading, updating or deletion (CRUD) of original data. Transactional data is also referred to as operational data. Analytical models encompass one or more copies of transactional data at some point after the original transaction occurred successfully. Master data models embody the source of truth for a given transaction, such that quality of the data (or metadata) can be ensured. The analytical database may be a production system, as well; however, its purpose is to provide access to one or more copies of transactional data.

Transacting Transactions with Transactional Databases

Attributes commonly associated with transactional databases include high performance and heavy writes. We think of a *transaction* as a unit of work conducted within a database such that a change to the state of the data is observable in real time. Often, transactional databases are associated with applications that are used to collect data directly about the world. Examples include front-end applications where a human or machine interacts with an application that creates or modifies data in some way. For many applications, a transaction must be guaranteed such that it is accurate and complete with data intact. In other words, each transaction must be ACID-compliant: **A**tomic—all or nothing, **C**onsistent—follows databases rules, **I**solated—secure and independently processed; and **D**urable—can overcome failure. ACID compliance is a requirement for any transaction to occur.

Operational Database/On-Line Transactional Processing (OLTP) Databases

Operational/On-Line Transactional Processing (OLTP) databases are data structures where data can be updated in real time. Typically, these are the data sources to the applications in which a user interacts with the underlying data structures. Historically, transactional databases have been structured (SQL), ACID compliant[26], and relational in nature. Relatively recently, NoSQL databases have gained popularity. These include document-oriented databases and key-value stores.

26 ACID means Atomicity, Consistency, Isolation, Durability.

This chart from a Quora article[27] by data engineer Chris Shrader provides a high-level summary of different types of databases.

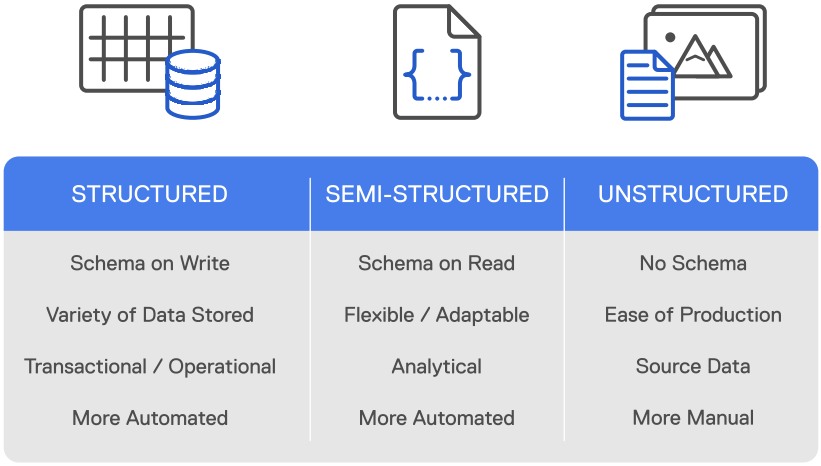


Figure 7. Types of Databases

Analytical Databases

Analytical databases allow users, services or applications to store and process data at some point after the initial collection of data from a transactional data store.

Online Analytical Processing (OLAP) is a type of database that allows users to analyze multidimensional data in an interactive way. This means that users of this data structure can aggregate data in more than one dimension (roll up data from small to large), drill down (go from whole data to more granular components) and manipulate the data to remove or move around data. Essentially, the OLAP data allows users to view data from different *dimensions*—meaning a way to view the facts of the database categorically.

27 https://www.quora.com/In-what-situations-should-one-use-a-given-database-such-as-MS-SQL-MySQL-NoSQL-MongoDB-or-GraphDB-over-another-What-are-the-scenarios-in-which-each-one-is-used-What-is-the-advantage-or-disadvantage-of-one-over-another

A **data warehouse,** illustrated in Figure 8, is a system used to analyze and report on data located in a centralized repository of disparate data sources. Data is copied from where it is created into the warehouse—mostly through an *Extract*, *Transform and Load (ETL)* process. Data from the warehouse is accessible through queries (such as SQL), which return values to the applications (or users who simply want to analyze the database directly).

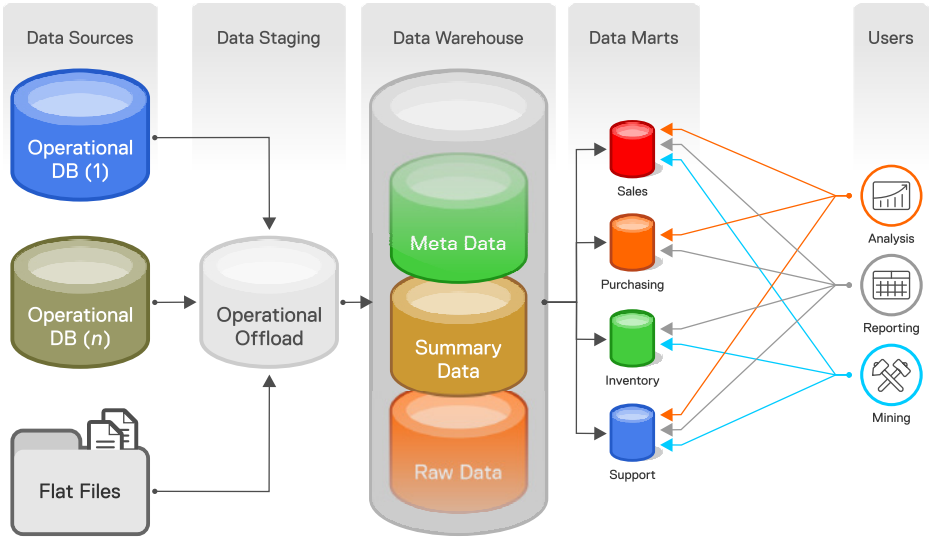


Figure 8. General data warehouse architecture

A **data mart**, shown in Figure 8, is a subset of data from the data warehouse concerned with the elements of data and data management important to a group of users (that is, department, line of business, etc.), within an organization.

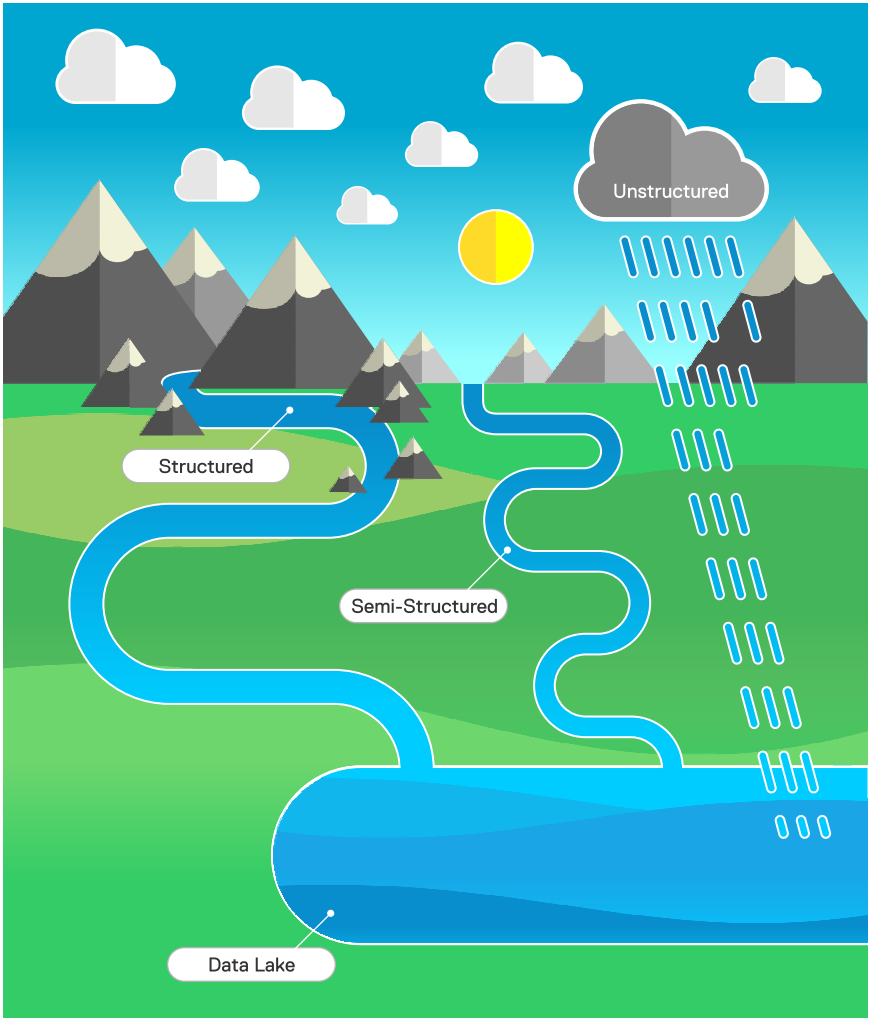


Figure 9. Metaphor of a Data Lake

A **data lake** (Figure 9) is a repository of data consisting of structured, semi-structured, and unstructured data. The mountains represent the rigidity of a schema; the shape of the river may change frequently as it cuts through the softer lowlands, but it is still a river; heavy rain clouds are unique and without defined structure.

Business intelligence is the strategy around a system of technologies to garner understanding about their business or operations through the analysis of data.

The data lake's vast repository of organizational-wide, raw information can be acquired, processed, analyzed and delivered. Unlike a data warehouse, which only consists of structured and semi-structured data, the data lake also includes unstructured data. Having all these datatypes in one place has enormous advantages. Schema-less data like email, image, video and document data make up most enterprise data. If each business unit within an enterprise puts its unstructured data, semi-structured and structured data in the same place, then the enterprise should be able to analyze all different types of data across the entire enterprise more easily. At best, clear inferences would be drawn against all datatypes organization-wide and presented in a context-appropriate way to individual business units. At worst, all the data is in one spot to break down data silos at the storage layer. In reality, most enterprise data lakes are somewhere in between. But even if data is centralized, most organizations do not know what data resides in the data lake, and they have a difficult time understanding the relationships that exist between the data in order to make sense of it.

In order to make data lakes valuable, especially across a multicloud landscape, it is essential to extend the value beyond a landing zone, as well as beyond a set of specific tools to derive value from the data. We must incorporate the notion of using endpoints that allow data to be consumed into the context of the calling application. We must include disparate processes and services from any cloud. We must know what data to access, how to access, consume and govern them. This is all done with a **data fabric**, a concept introduced in Chapter 1.

A data fabric extends the notion of a data lake such that the data can be available between connected endpoints and distributed across one, more, many or any clouds. Data can flow from source to target in batch or real-time over the same, idempotent pipelines. Any datatype can be accessible to any authorized caller, manipulated to the right format, contextualized to the expected ontology, and governed by the appropriate policy. A data fabric is not a place where data rests, rather it is an interconnected framework of nodes and edges that allow the data to be active and alive.

Integrating the Data Fabric

If integrating application data were easy, anyone could do it. Applications are written in a variety of languages using a panoply of libraries. This makes it hard for them to talk to each other. Fortunately, specifications for various types of programs have made the application communication process easier than ever. *Middleware*, the layer of software that translates between applications, has been used for decades. From coding languages to document formatting and all sorts of filtering functions, middleware has been the mortar to hold together the foundation of enterprise applications. For years, many middleware products have been customized to fit the needs of a given enterprise's

application stack. Companies could take months or years to write code from various middleware platforms, just to have their applications transmit data to and from each other. Middleware was seen by many as a "necessary evil," but that changed not long ago.

Little more than a decade ago, a software integrator named Boomi (which means "*Earth*" in Hindi) decided to change its business model to leverage what was, at the time, a new concept: the cloud. Startup CRM Software-as-a-Service (SaaS) companies like NetSuite and Salesforce were trying to replace cumbersome software. They needed a middleware platform to help them integrate with customer applications. Boomi was one of, if not the first, integrator to compete. Boomi created a software platform that allowed its users to design integrations in the cloud, while allowing for runtimes to deploy anywhere. Additionally, the platform provided a simple approach to integration, and required no coding to integrate. This gave birth to a new industry, now dubbed *Enterprise integration Platform as a Service (EiPaaS).*

Boomi was and continues to be highly successful. Boomi removes from the customer the burden of building and maintaining physical hardware merely to support integration services, multi-step upgrades, and costs, while increasing speed to value without the need to be able to write code. Boomi was quickly acquired by Dell Technologies and was one of the first companies that leveraged the cloud to disrupt a long-standing market. It continues to innovate and lead in its space and continues to grow at the rate of six new customers a day.

Even with the success that Dell Boomi and the iPaaS market have provided customers worldwide, there is still much disruption left to occur. But it always starts with understanding the different types of architectural patterns that support the use cases for each integration process. The patterns from the input and output pipelines can support any of the patterns you will read about in the next section, but within the processing core, there are two prevailing patterns that are pervasive with a variety of middleware today.

In the next section, we describe several of the most common integration patterns.

Common Integration Patterns

Point-to-Point

Figure 10 illustrates a point-to-point integration pattern, or *singleton*. Point-to-point is a common type of data access architecture where one source and another target are interconnected through logic and mapping. This is a process where synchronous or asynchronous data transfers occur based on the rules set up between the two endpoints.

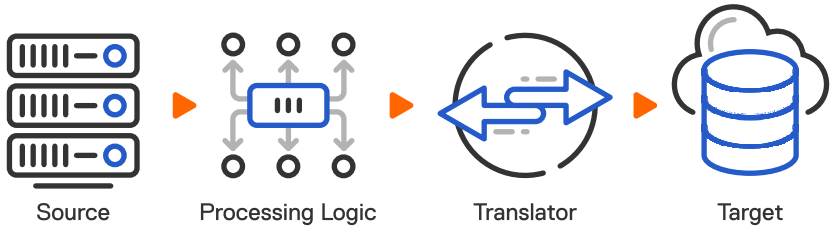


Figure 10. Point-to-point integration pattern

Pub-Sub

A publish and subscribe (pub-sub) queue provides a way for a message *broker* to receive an incoming message request such that other applications can subscribe to the queue—called a topic or a *stream*—so that other applications can *subscribe* to the topic or stream. This pattern is *asynchronous* and can have various semantics associated with it, such as *guaranteed delivery*, *exactly once, and fire and forget*. When conceptualizing data streams, it is helpful to think of pub-sub. Unlike traditional pub/sub, however, modern data streams can handle datatypes other than messages. In addition to messages, a modern streaming engine can handle batches, events (state changes), append-only logs, and even database transactions.

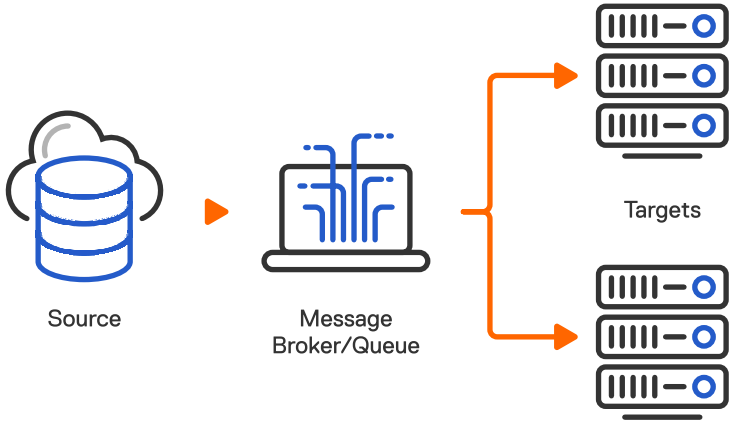


Figure 11. Pub-sub queue

API/Web Services (Request-Response)

Application Programming Interfaces (APIs) are request-response tools for interacting with an application. There are several different kinds of APIs available. Web APIs are what come to mind when hearing the term *API*, but operating system, database, hardware and software library APIs are also available. APIs are designed for users to request data and access services or functions without having to know how to write code in the application language or import into code-specific libraries of the host application. APIs provide a layer of abstraction from programs to deterministically grab resources. And these deterministic facilities—that is, the expected behavior—are often called *contracts*.

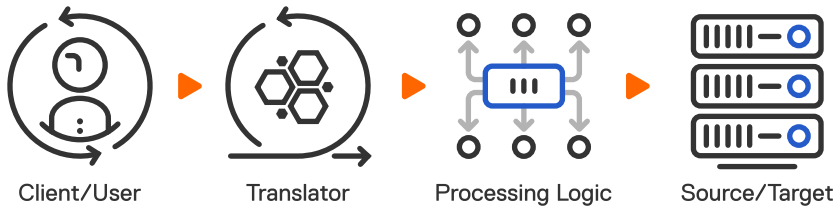


Figure 12. API/Web services (request-response) example

Simplicity is at the heart of using an API. If your application requires data from another application, simply make a call to that endpoint, using the documented procedure of the other application, and the other application responds with the requested data or service. If you ask for information, assuming you are authorized to receive a response, you get a response. Further, developers know what the expected behavior of a given call is. This makes application communication easier than it would otherwise be without APIs.

Note that authorization is required for the interaction to work. This is a key part of APIs, or at least API management. API management provides a mediation layer between caller and responder. An API management layer can involve a multifaceted approach to governance. This may include throttling, rate limiting, access mitigation, and a variety of other services to ensure that users get what they need and nothing more.

Extract Transform Load (ETL) {alternatively ELT}

Extract Transform Load (ETL) is a bounded stream. This means that a start and stop *offset* (time, break point) is known of a given series of data. Typically, data is bound by time, meaning that its sequence is measurable in time. Also, large volumes of data are copied from one data source to another, then translated in a *processing layer* so that the target application can use the target data store as its pristine data source. ETL is associated with batching. And batching is associated with data lakes, data warehouses and data marts.

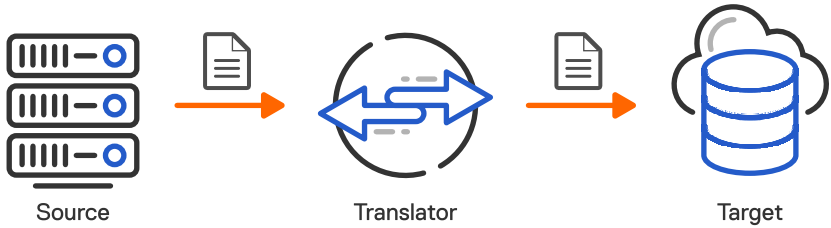


Figure 13. ETL example

Although there is a lot to consider with each of these types of repositories, the fundamental notion is that these are the places where data is analyzed after it has been collected in its raw formats and converted to a format that is easier to manipulate.

CHAPTER 4

Input and Output Pipelines

Where Are the Data Sources?

At first glance, it may seem that we would want *all* the data. After all, who knows what kinds of insights we may find when correlating clickstream data and the number of times someone turns a light off or on in a broom closet? Perhaps this is a far-fetched example; but the question is valid. Shouldn't we save all the data, and derive meaning later?

Perhaps. Given that Dell Technologies is a leading provider of data storage, you might expect us to advise, "Yes! Save all the data!" Yet we know that saving all data is not practical or tenable. It could even be a violation of corporate and legal policy to keep certain data or to keep data indefinitely. There are myriad technical practicalities beyond governance issues that may affect the company, including bandwidth limitations and incorrect, stale, corrupted, personal or superfluous data.

For example, for aggregated data from Internet of Things (IoT, or Edge devices), it may make sense to simply limit traffic by sifting out only the important bits of data from the rest. Perhaps WAN bandwidth is limited. Transmitting anything other than anomalies—or signal—may add traffic or latency on an already overtaxed network. Also, having excess noise in a data pipeline may create obstacles to achieving a desired outcome. Keep it simple. Send the data into a pipeline necessary to achieve the desired outcome and nothing more.

Figure 14 illustrates an input pipeline for an IoT (Edge) use case, where the organization has options to send all or some targeted data into a pipeline for processing. Targeted data incorporates listener APIs to establish a data relay to the processing core if, and only if, certain conditions or rules are invoked to trigger such an event, a state change within the data.

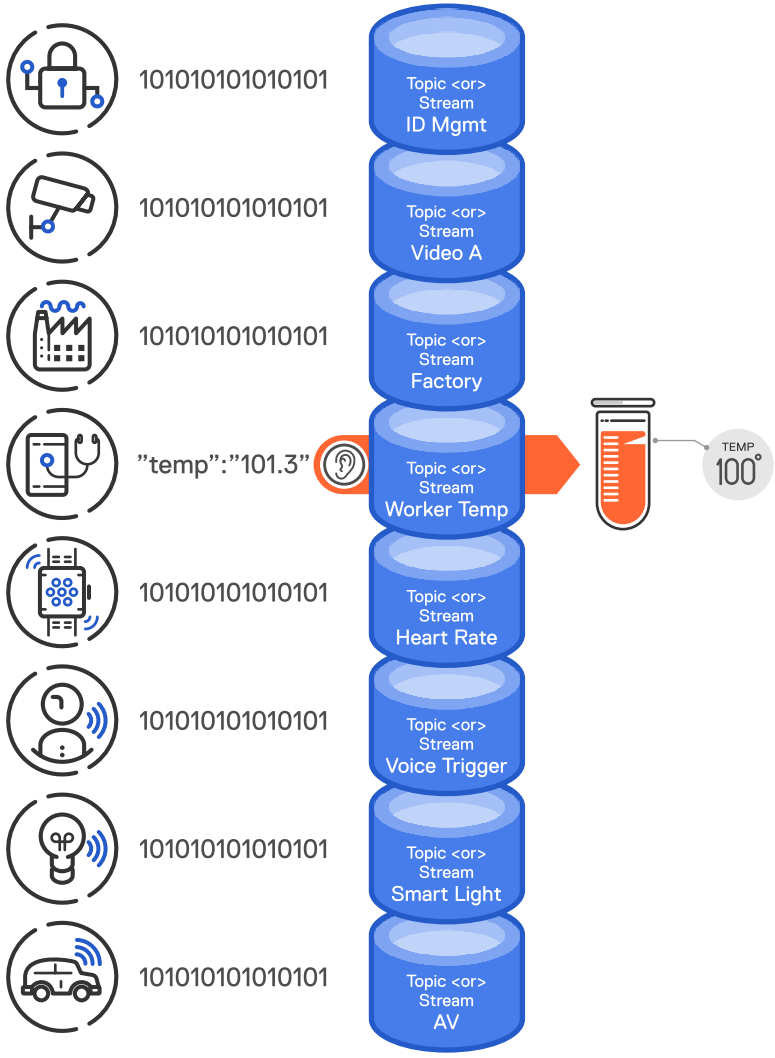


Figure 14. Internet of Things (IoT) use case

Keep in mind that the Edge use case example references one of many similar input pipeline patterns. As applications get smaller and look more like nanoservices, the network chatter between one service and the next resembles an IoT pattern. We often advise organizations to treat all data sources and pipelines more like IoT patterns, remembering that bigger data comes from smaller apps. Apps will continue to get smaller and, as they do, more sources of data will emerge, especially for business.

Even we humans are becoming more like edge devices ourselves. With our myriad wearables and hundreds of apps and services on our smartphones, data about us is being aggregated at our personal gateway (smartphone) and processed locally, or sent to an app's server at a core or cloud location for analysis. It will take bandwidth and a better way to store data to deal with the accumulation of transmitted signals and their data concomitants. Whether we are talking about endpoints for Edge, SaaS, web applications, robots, or smart cities, all data patterns from the input pipeline are similar. Listening for the source data from these endpoints to achieve a desired outcome is the critical first step to building modern enterprise data pipelines.

Governance

It's 3 AM. Do You Know Where Your Data Are?

Governance is an overworked term whose meaning may vary from organization to organization. What is important about governance to a sole proprietor of a pizza shop in Alaska can be vastly different from what matters to the executive board of an enterprise in Poland. Regardless, there are principles of keeping track of data to which each of these companies must adhere. We only touch briefly on governance here, a complex concept that is the subject of DAMA's DMBOK[28]. A pragmatic approach to governance includes pipeline layers between physical or logical infrastructure and the desired outcome of the business or consumer. For this reason, we provide you with a conceptual map of a pragmatic enterprise data strategy. From infrastructure to desired outcome and user experience, layers of governance are needed for each pipeline. Pipeline layers include data assembly, patterns, services and controls. Governance can be applied in any form at every level of a pipeline's connected endpoints. One such area is at a data assembly layer.

28 Data Management Association (DAMA): Data Management Body of Knowledge (DMBOK)

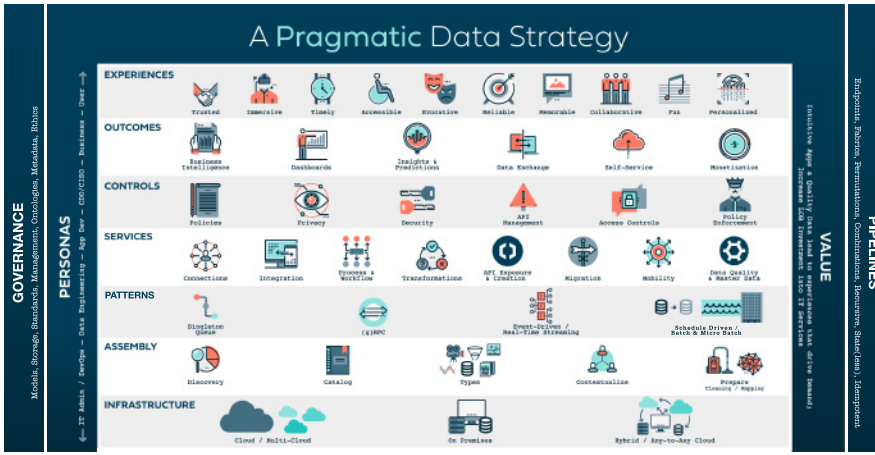


Figure 15. A pragmatic data strategy

Can you imagine reading a book (other than a novel) without an index or a table of contents? Such books exist; but unless you've read the whole book cover-to-cover (and comprehended it), you cannot possibly know what the book is about, or the salient points stated within it. Sure, you could trust the word of a friend or a critic; but then you're reliant upon *their* version of the truth. For those who remember card catalogs in a library, it would be next to impossible to find a book in a library unless you knew how to look up the title of a book with some method. Yes, a librarian could assist you with an inquiry; but what happens if they are on a break or helping someone else? Or can you imagine shopping online with no pictures of the items you wanted to buy? No textual description or information about item size, color, or price? Indeed, you could have pictures; but if there were no context, would you know what you were buying, let alone what you were looking at? Would you buy it? Would you trust the website? For this reason, there must be contextual elements wrapped around the data in such a way that it can be discoverable, locatable, indexed, and searchable. If any of these functions are absent, then governance becomes elusive.

Before any policies can be implemented on data, *assembly* is needed. By assembly, we mean defining where the data are, what the (meta)data are, who or what will consume the data, and how the data needs to be presented, pre-processed or staged to the next leg of the pipeline. It is through assembly that data is discovered, cataloged, and prepared for the rest of the journey through the pipeline. Data in a pipeline can become governable provided it can be found. The way to (easily) find data is with a *data catalog*.

No Catalog? No Data. Know Catalog, Know Data.

A data catalog provides an effective way to govern data. It also is the foundation of effective data management as it establishes a trustworthy start to the rest of the data pipeline. Knowing where to locate data is critical to make certain that they are correct and accessible—whether available to everyone or to only those parties who are authorized to view them, and no others.

A data catalog provides a platform for consumers—customers, citizens, developers, stewards etc.—to know precisely where to find the data that must be pipelined to the applications that they're using or building, and the locations to which they will need to persist transactions. Locating data also means that user data or metadata can be enriched as part of a pipelining process that may be consumed by another application, process or service. The actual catalog could be—and arguably *should be*—turned into a service in its own right. Making the catalog a callable service endpoint fosters reusability and simplifies pipeline searches for another important step, preparation and cleansing.

Preparing and cleaning data becomes a welcome outcome from a data catalog as well. Data scientists spend a lot of time preparing their data for analysis. It is imperative that data be clean and ready for consumption from a myriad of machine or deep learning models into which pristine data is loaded. Data should also be enriched or filtered, which means that the stored bits can be cleaned so that they can be analyzed better.

Clean data ultimately leads to better ways to protect privacy and secure data resources. This can be a boon for regulatory compliance and risk management as well. The General Data Privacy Regulations (GDPR) implemented in Europe affect every business or government that conducts business in Europe. Even if your organization conducts business only with non-European companies and citizens, what happens if your *website* happens to be hosted (in whole or in part) in a European country? If you have cookies that collect data of any kind, even if you do not exchange money with a consumer, you still may be subject to EU regulations and sanctions.[29] What about personally identifiable information (PII), the Health Insurance Portability and Accountability Act (HIPAA), and a host of other regulations? If you don't know where to find the data, how do you know how great your exposure to risk is?

When data is corralled, then data sources can be accessible to all of those who need it. Lines of business and the analysts that monitor data within them, data scientists, developers and even customers can take advantage of the data that is made available to them. Each line of business within the enterprise is looking out for its interests. Each business has its own analysts combing over details about data—old and new—about the buying patterns and behavior of the customers that purchase and use products or services of the company.

29 https://www.dickinson-wright.com/news-alerts/what-usbased-companies-need-to-know

APIs: Consume Data and Make Data Consumable

Application Programming Interfaces (APIs) are constructs that simplify access to data, functions and services. Due to the sheer volume of applications that are Internet-based, APIs are associated commonly with web applications today. We will refer to web APIs as *APIs*. After all, when we think of a multicloud architecture, we access data, services and functions in the cloud. Thus, it makes sense to consider of APIs in terms of web APIs.

As described above, web APIs are an integration pattern that connects any separate parts of an application, or any different applications, together.[30] APIs are abstracts of functions or services that are packaged in a clearly defined way, such that when a program follows the rules of calling the API, it receives an expected result from the API. The expected result from an API call is often referred to as a *contract.* And a program can consist of—in whole or in part—one or more API calls put together in a meaningful way. Additionally, APIs exist so that developers do not have to write code into their application that has already been written.

As discussed earlier in this book, applications—certainly Estate 2 applications—can be composites of functions and services from multiple clouds. This means that a function in hyperscaler *p* may need to post data to a service within hyperscaler *q*. The application developer writing the program *could* elect to import libraries into the code, spend countless hours trying to understand how different functions or methods work, and write dozens or hundreds of lines of code to create main functions, helper functions, constructors, classes and interfaces, so that parts of the code could call other parts of the code. Or, the application developer can simply look up the web API and allow the developer who wrote the API code abstraction to take care of all the details. The application developer may not know *how* it works line-by-line, but does know exactly what to expect when making a call. This saves a good deal of development time.

Extracting data from third-party data sources was once a difficult undertaking, but with web services and exposed APIs, the difficulty of extracting data is greatly reduced. From an input pipeline, using APIs to access third-party data is a viable way to inject disparate sources of data into the data pipeline to process data further.

30 https://hackernoon.com/what-are-web-apis-c74053fa4072

Multicloud Data Security and Privacy

It should be obvious that data security and privacy in a cloud computing environment are important for a host of reasons. These reasons are well documented in the literature, and we won't delve into this topic to a great degree here, as we can devote an entire book to this topic alone. Make no mistake, we take security and privacy very seriously and so should any customer adopting a multicloud data strategy. Instead of talking about all of the possible options to address security and privacy, we will summarize three areas for now: security between integrated endpoints, authorization to access data through an API mediation layer, and governance for data privacy.

As we stated in the first section of the book, processes or composite applications built from services or functions from different clouds are becoming a standard. Data is easier to access than ever, and it likely lives in far more places than we probably realize. This is a blessing and a curse. The blessings are obvious. Access to data wherever it is can enrich one application's utility while, in turn, become a way to monetize its now enriched data to other applications. In other words, data can be curated by one application from another, processed, and then monetized. The curse is that all these accessible endpoints can become potential vectors of security risk or privacy violations if not controlled.

Authentication

Processes or applications built across multiple clouds mean that authentication procedures need to be considered for efficient data pipeline design. Authentication methods for one cloud-based service may vary greatly from another. Decisions to house, cache, tokenize, encrypt, protect and perform other procedures on certificates and keys needs to be considered, not only to protect the application and data, but to also ensure that the process is not unnecessarily wasting precious time amid calls between endpoints at run-time. Furthermore, if data, application functions and authentication services are not co-located in proximity to each other, or if endpoints are in geographically disparate locations, then latency could end up making the application slow, or worse, unusable.

Some multicloud strategies require that no internal data is accessible within public clouds. In other words, stateless applications may be located in public clouds; but no data may persist there—only behind the firewall. Alternatively, other strategies permit data to be located in a cloud environment, provided the customer can extend the firewall *into* the cloud (as a virtual private cloud). Either way, protecting the efficiency of endpoint-to-endpoint authentication to access data within a multicloud architecture requires a runtime engine that is flexible and can be deployed anywhere, based on an organization's specific security requirements.

Authorization

Once authenticated, authorization is the next security challenge to address. Accessing data can become problematic, especially when one service's output is based on another's input. This, too, can incur latency; but latency plays a secondary role to another, bigger problem: weak mediation. When designing multicloud applications or processes it is critical to place restrictions on what data can be accessed, how much data can be accessed, by whom, at what speed, for what duration, and to limit the quantity of requests. These practices are important especially when making data accessible to callers outside of the organization. If throttles are not put in place to mediate the access of data, problems are likely to unfold. Some undesirable outcomes include, but are not limited to, not fulfilling monetized potential, a caller's ability to access all possible data, limit others' access to data, flood the network with requests (DDoS), or worse.

Authorization safeguards can be implemented through an API management strategy. Through this strategy, caller access can be governed, data available for consumption can be controlled, and callers can be tracked and managed so that unauthorized access is avoided.

Privacy

After considering security around authentication and authorization, focus on privacy must be taken seriously. Violation of rules like the General Data Protection Regulation (GDPR) or the California Consumer Privacy Act (CCPA) come with stiff penalties. These laws are meant to protect the data privacy of individuals. Companies conducting business in countries where these rules or others like them apply, must adhere to the law and, therefore, must embed into their data architecture ways to catch personally identifiable information (PII).

The problem is that governance around PII is very difficult to achieve. In fact, it is nearly impossible to achieve if your data architecture strategy contains no way to discover data so that it can be governed in the first place. Some safeguards that are required for good governance include ways to find data that *could* be personally identifiable. Next, it is important to catalog (or index) the PII data in a way that any business context can be aware that data *is* private. This would include information like full names or addresses, social security numbers, credit card numbers, etc. Some PII data may be easy to discern if they are in a database with a clearly labeled schema; but what about data found within unstructured data, or injected as erroneous diacritics (99`2_34-1111) between numbers that would otherwise be easy to identify, or numbers-as-text (one-four-three…)? Once found, the data needs to be prepared (cleaned), and put into some database in order to become a source of truth against which other data can be compared. And this is one aspect of the governance process. There are other steps that we are omitting here; but you can see that protecting the privacy of users is no easy feat.

Think that privacy laws don't apply to you because your organization does not do business in one of these countries? If you have a *Cloud-first*, *All-cloud* or *Multicloud* architectural strategy, can you prove that your data isn't physically taking residence in a data center housed in one of these countries? If you can *prove* that your data *isn't* in such a country, good for you. Otherwise, you may want to either invest in a governance strategy, with which Dell Technologies can help, or put all of your systems safely on-premises (we can help there, too).

While this book isn't meant to cover the full gamut of data security and privacy, these two areas of a multicloud architecture must be a part of your strategy. Data security and privacy protection are becoming more important for the future development of cloud computing technology in government, industry, and business. Data security and privacy protection issues are relevant to both hardware and software in a multicloud architecture.

Events and Streams

Streaming is uniquely suited for logs—every click, every play, save, and "like." Every action a user takes within the application is *logged* and processed for immediate analysis. Log analysis is one example of how streaming services are changing the way that data is harvested and used for analysis and future platform/product development. Analyzing log streams is not only good for improving products and services, it is good for monetization as well.

As a modern data pipeline, streams provide an organizational system of data flow similarly to how the central nervous system from which applications source their data.

Streaming is an overworked term which can invoke thoughts of telecommunications: "the act of sending sound or video to a computer, mobile phone, etc. directly from the Internet so that it does not need to be downloaded and saved first."[31] Digital audio and video transmission is not what is meant by streaming, though the process is fundamentally the same with logs, considering state transitions are recorded serially. Additionally, data, as well as logs, is stream-worthy. And data—which does include audio and video transmissions among other types of data (for example, files or messages)—can be buffered continuously. Unlike a film or a sound clip, a log is written (typically) in text, and a streaming log has an indefinitely open channel of communication that can be analyzed "in-flight" by one or more additional systems. Through this analysis, derivative streams can be spawned based on certain events, or state changes, that occur within the originating stream. When new information is recorded to the primary stream, the transmission is often noted with an ordinal stamp, along with other stateful information. Imagine if you went to a theater to view a film that never ended!

31 https://dictionary.cambridge.org/dictionary/english/streaming

Streaming data is any data currently transmitting in a serial fashion as events occur.[32] In other words, data streams are append-only records of state changes that occur for a given system or process. We will discuss streaming in greater detail, but first, let's understand the importance of logs.

It's All About Logs

Applications, systems, services, and devices have logs. For a given application, a write-ahead, replication, and event log are common. A write-ahead log is piped to memory before I/O is committed to disk. A replication log records data about replica states to multiple locations before requests are acknowledged. Monitoring logs are created so that changes of state can be analyzed. There can be other logs, as well; but rather than discussing different types of logs in detail, we will address their importance instead.

Logs record to an append-only file every database *event* or state change. Depending on the purpose of a log, it can be used for humans (and non-humans) to troubleshoot, extract meaning from the information collected, replay the same data structure elsewhere, or do something useful—like perform analysis with the insights gathered from the logs.

Figure 16 presents an image of the processing core involving a pre-processed durable buffer to maintain state before processing, and a buffer to hold the state of processed data. Logs have been used for quite a long time, so why focus on them now? We will explore this; first an anecdote.

32 Aragues [2018]

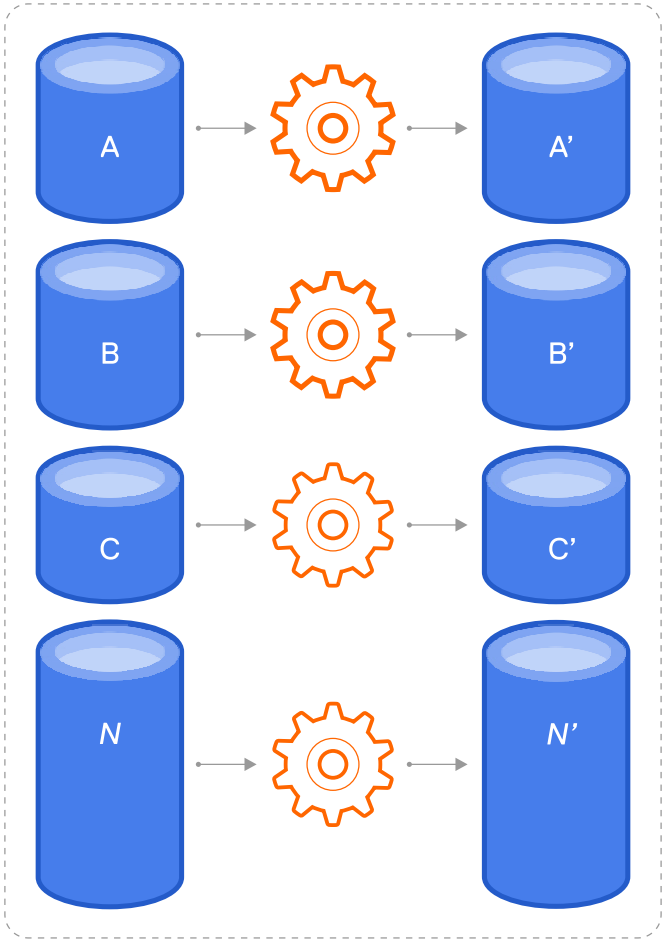


Figure 16. Source to derivative pipeline

How do you tell if your website is under a Distributed Denial of Service (DDoS) attack? Did the hackers notify your team? Does your company have a fortified moat or honeypot that immediately thwarts the intruders? Is your Security or Network Operation Center (SOC or NOC) staff sophisticated (or clairvoyant) enough to preempt the incursion? Or do your customers begin calling anyone in your firm that can be reached? By phone? Knowing about a DDoS attack almost always happens *after* the attack begins. Sometimes minutes afterward; hours even—or more. Logs will be analyzed to be sure. And those logs provide critical information as to which systems are down, where, when, and much more.

Logs are extracted from different systems—often manually. In the DDoS use case, logs would be used to understand the compromised attack surfaces.

Logs are retrieved from what could be every one of your systems touching the public Internet (and more). Logs are compiled, collated, cross-referenced and, ultimately, analyzed. And the process repeats until steps are taken to get the systems back online. This procedure can take time. But each time a log file is batched, there is stale data. Of course, having batched logs provides clues as to what *happened,* and that is helpful. Wouldn't it be helpful if logs could be monitored and analyzed in real time? And not only for DDoS. What if logs could be analyzed for every application? Every system? On-premises or in a hyper-scaler cloud?

Turns out, you *can* analyze logs in real time. It's not only possible; but most customers who are adopting it see its potential in more ways than anticipating attacks. When examining one-to-many (1:M) data transmission patterns, the ubiquity of the use cases becomes apparent.

Figure 17 shows a listing of the typical operations of various processing engines. Some processing engines can perform more operations than others. Processing engines can embody SQL databases, Hadoop clusters, data lakes or warehouses, or real-time streaming engines. The term "operation" used here denotes transformations and functions that can perform one or more of the actions listed. This list is not exhaustive.

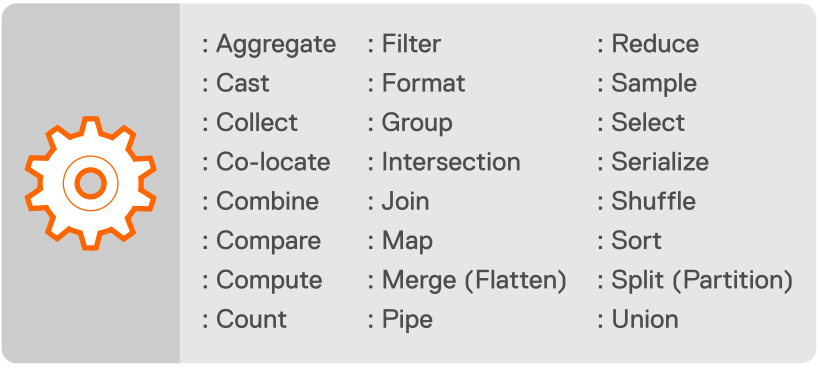


Figure 17. Some typical operations

Output Pipelines: Configure the Outcome

Your credit card was stolen, now what? You requested a ride from your favorite ride-sharing app. How does the driver get all the data she needs to retrieve you? You're interfacing with a chatbot about your deplorable Wi-Fi service on the plane. How does "Joe" know whether to issue you a credit for

your spotty Wi-Fi? Answers to these questions and many others are provided within the processing core.

Source data fed into a durable buffer by the left-most (blue) pipes are processed by the streaming engine (red pipe, below). When data is processed, then it is placed into a derivative (blue pipe) buffer, from which it is accessed by one or more applications or processing pipes. There may be a series of blue, red and blue pipes before data processing is completed such that it is ready for dequeue by the applications or services upon which it is dependent.

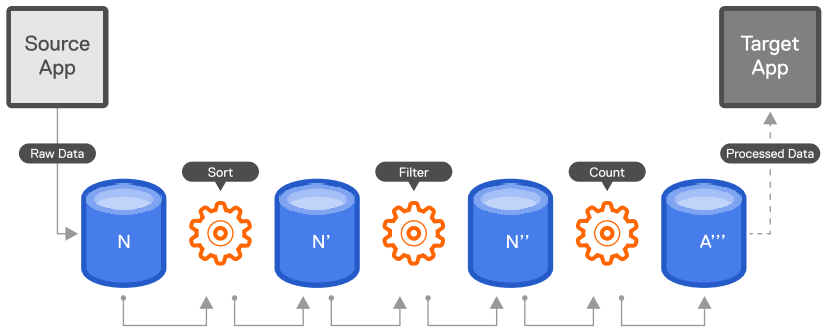


Figure 18. Series of processing steps between raw data and final output

This type of scenario does not have to involve one and only one type of persistent buffer or analytical engine. For instance, a given process could include a topic or stream (a buffer) as the pre-process layer for data writes. An API connects the topic to which the producer wrote the payload and is consumed and processed by a Spark streaming engine. The processed data can then be written to a Pravega buffer, where data are subsequently pipelined into a Flink job to join a data source to the processed stream, then map and filter it so that the product which will ultimately end up publishing data to some other queue.

Keep in mind that the previous pipeline is an example of a possible use of multiple buffers. In truth, any combination of durable buffer or processing engine could be used at any sequence of the data pipeline. This "process-final" data becomes the data source for applications or services to consume. The blue pipe is the source for the red pipe, which writes to a blue destination; and the process may continue if necessary in order to feed an application the resultant processed data upon which it relies.

Additionally, a data pipeline can also include one or more sequences of red pipelines before final write to the target blue data store. Each of the processing functions within the red pipe can be as simple or as complex as the developer chooses based on the specification of the business.

As an example of orchestrating an outcome, imagine that a cellular service provider offers a service to notify customers of inclement weather via SMS text. First, data must be sourced and staged to get processed.

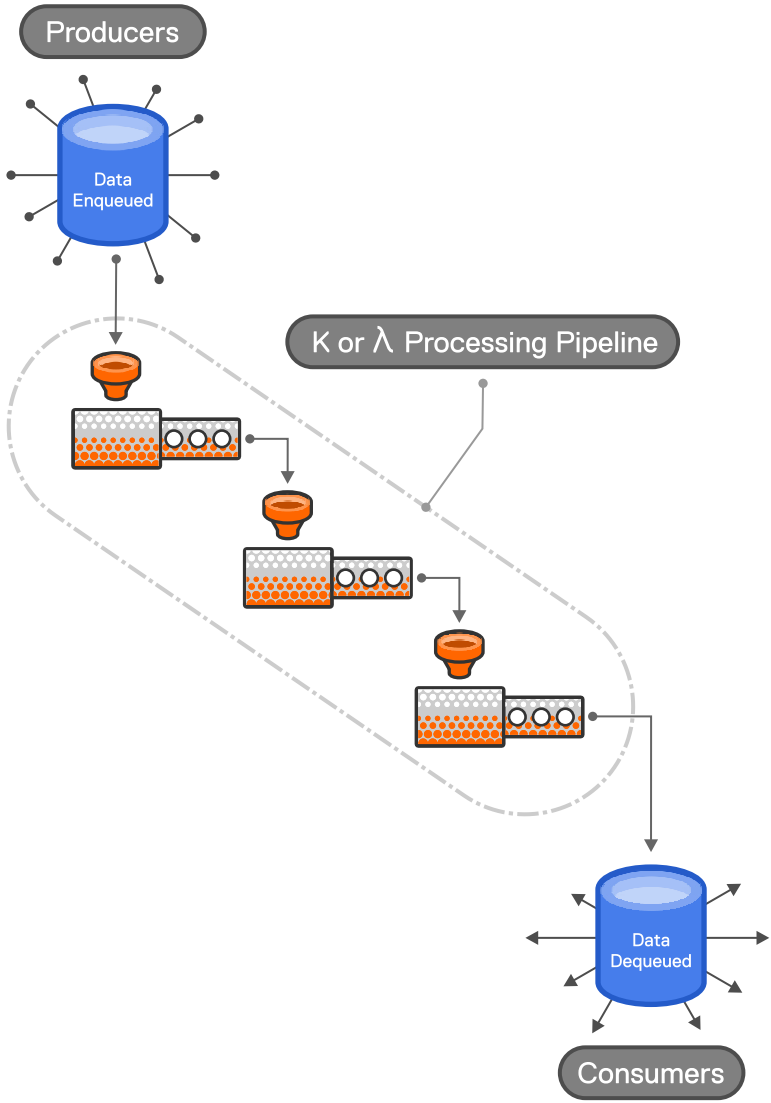


Figure 19. Inside the processing engines

Raw data enqueued into a persistent buffer is piped into a processing engine for computations. Once processing completes, post-process data writes to a derivative buffer which can be dequeued by applications, services or functions.

Suppose that data is gathered from a few raw feeds: weather websites (via API call), national newspapers (RSS feeds), and a stream of tweets (#weather). Each raw data feed is published (written) to a staging blue pipe. From here the data can be merged into one or more processing queues wherein data are mapped, reduced, joined, sorted, and trimmed. Finally, the processed data are written to a durable buffer to which the notification service subscribes. From here, all users of the cellular service receive a message. And that message is *"Snow Squall Warning til 3:45 PM EST."*[33]

This is an example of a simple outcome orchestration. Multiple data sources were gathered, the raw data processed and, finally, the data written to the data store of the server hosting the messaging application. From here the message is broadcasted to the cellular provider's installed client messaging applications installed on each customer's cell phone. It took an entire data pipeline to present the message to achieve an outcome; and the outcome is where the output pipeline comes into play.

The output pipeline is concerned with the post-processed data. Typically, this means that the data will land in a persistent store of some kind; but this is not necessarily a requirement. There are cases where data can flow from source to target with no persistence in between. At some point, data will be stored somewhere, if only ephemerally, as in memory, or cached. To keep things simple for now, the messaging app can be a straight feed from a set of data sources, independent of techniques to temporarily store data. But many applications use persistent storage of some kind for various reasons, including backup and recovery, log roll backs, ancillary data feeds, or one-to-many data store.

Figure 20 shows an example of an incomplete outcome. One missing critical piece of information to the consumer is the location of the icy conditions. There are many other pieces of data that would make the outcome, and the customer experience, better.

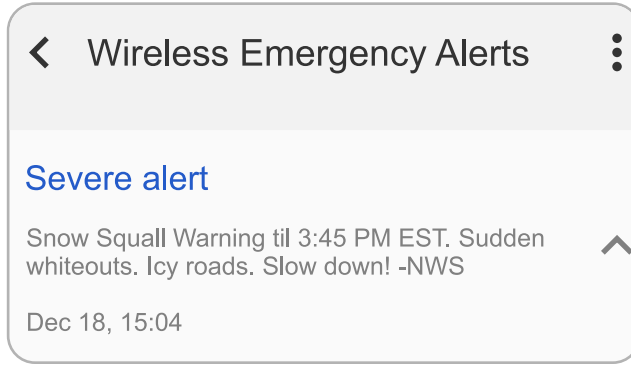


Figure 20. Example of an incomplete outcome

33 Actual message from a cell carrier on 18 December 2019.

The most important point about the output pipeline architecture is not whether the data is persisted, but the nature of the desired outcome in the first place. The most important part about input and output pipelines is not the methods used to get the data from one end of the pipe to the other, it's the viability of achieving the desired outcome itself.

It's All About the Outcome

Keeping in mind what outcomes we want the data pipeline to enable is critical to success. Unless research is conducted to explore the art of the possible and push the boundaries of new technologies, production applications exist for a reason. Production-level enterprise applications are often built to get a better sense of how products and services are used, purchased, and sold, how people interact with other people and objects, or how they explore their surroundings. In our observation, outcomes can be distilled into three general categories: (1) make or save (money); (2) produce or consume (things); and (3) reduce (risk). Arguably, you could *choose* to increase risk under certain circumstances, but that is typically a calculated trade-off for one of the other objectives, not a desired outcome. Additionally, the patterns are remarkably simple to model if the objective is clear. The clearer the objective, the easier it is to architect the pipeline to achieve that objective.

In our incomplete messaging example, the objective is clear: Produce helpful information. But how is information regarding a snow squall helpful to someone living in Phoenix, AZ, or Nairobi, Kenya? A complete outcome answers the question, *How do we make data that is important to specific users, and no others*? Setting up opt-in channels is cumbersome for the enterprise and annoying to the user. Even if your application provides hashtag subscriptions for various channels, users with high subscription counts may be desensitized to the myriad updates rolling through their feed. Conversely, users who subscribe to few channels will be unaware of any messages produced on subscription channels.

So in this example, how can we orchestrate an outcome that is better than a blanket broadcast of one weather anomaly? We can start by restating the question:

How can we produce important weather messages to our customers in affected areas?

Although we can't prescribe a specific procedure, what we *can* advise is that building pipelines to achieve such a clearly-defined outcome quickly becomes transparent, revealing the key questions. What constitutes an *affected area*? How should the user interface with the message? *What* should the message say and *how* should it be said? Once these basic questions are asked, the pipeline is easy to architect.

The enterprise business application owner can research available data sources. Perhaps some data sources are available currently and provide the necessary data; perhaps more are required. The business can consult with other business application or integration teams within enterprise IT to see if consumable data repositories or APIs currently exist and if so, perhaps the business unit can become a consumer of such resources. If not, enlisting outside resources becomes an option.

Once data resources are defined, then data extraction will become important, as only a subset of the data will be important for consumption. Conversely, enrichment and other data processing may be required, as not enough information is available through just one data source. In any case, your enterprise IT integration services or strategic integrators will be able to assist with this. Simpler still, self-service tools provided to the business by IT/IS can provide all the requisite tooling needed for your application team to build processed data. There are many ways to operationalize the pipeline to achieve the desired outcome.

In another example, your customer success/service organization might want a "360-degree view" of their customers. What customer service agent *doesn't* want a complete picture of their customer's environment, especially when they need to troubleshoot a problem? But while having a 360-degree customer view is a worthy goal, this in itself is not an outcome. The outcome is what the 360-degree view enables. For many customer service and success organizations, a 360 view is a needed component of minimizing resolution time to retain happy customers. Minimizing resolution time and retaining happy customers is the outcome.

Figure 21 illustrates a simple workflow model to achieve the desired outcome. From here, various teams who own different areas of the input and output pipeline architectures have a clear outcome and workflow. This kind of clarity is half the battle. Building the pipeline becomes the domain of the experts in IT, including DevOps, data-ops/data science, application development, and the app and data integration teams. These teams will use the tools and resources that are available to them, and the speed with which they can execute the formulation of the system, to help the business achieve its outcome depends on the tools and skills at their disposal. More on this is covered in Chapter 6.

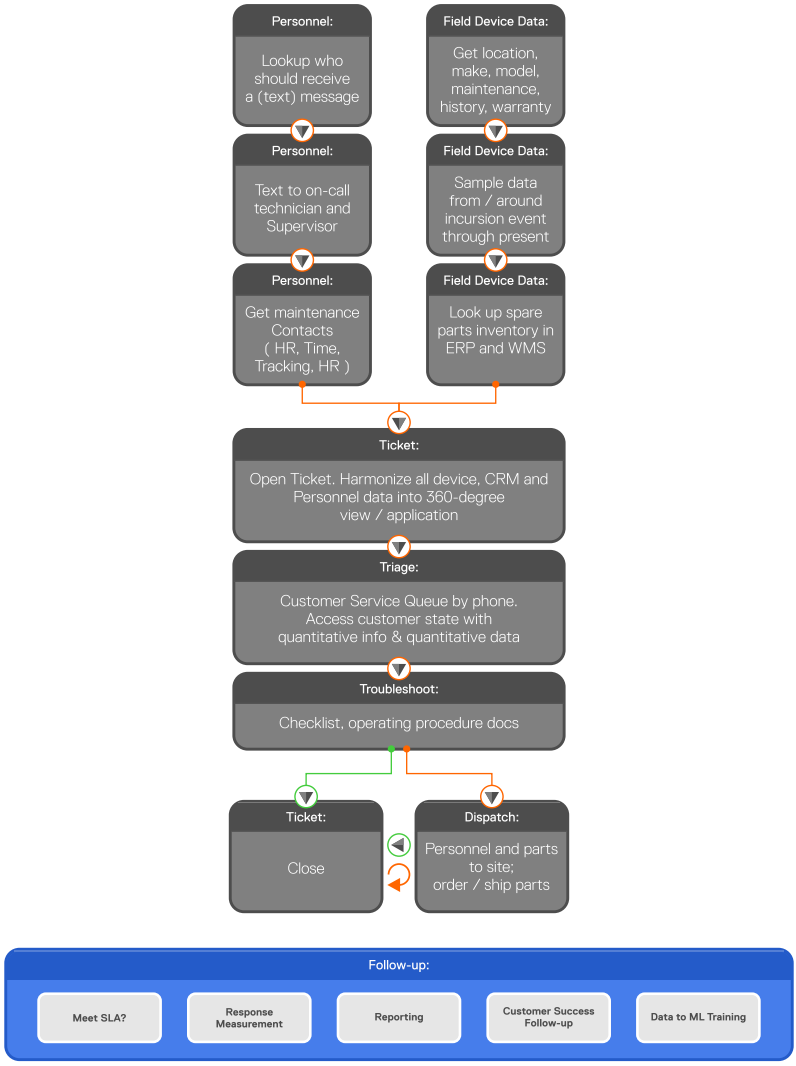


Figure 21. Example of an outcome orchestration, integrating people, process, and data

APIs for Outcomes

Use APIs to *your* advantage. Indeed, APIs can be the mechanism to orchestrate an outcome to reduce development time (save money) or expose APIs to external sources to make money. If you want to expose data to callers inside or outside of your organization, this is possible. Further, you could govern calls to access any data that you have (typically after it's been processed), so that you can profit from it. This is conducted through a mediation layer

referred to as API Management. API Management allows you to govern caller access to the data that you choose to expose through your APIs. This governance includes how many calls can be made within a specific period (rate limiting); who can access the data; how much data can be abstracted, and more. It is common for enterprises to charge fees to access data that they create or curate.

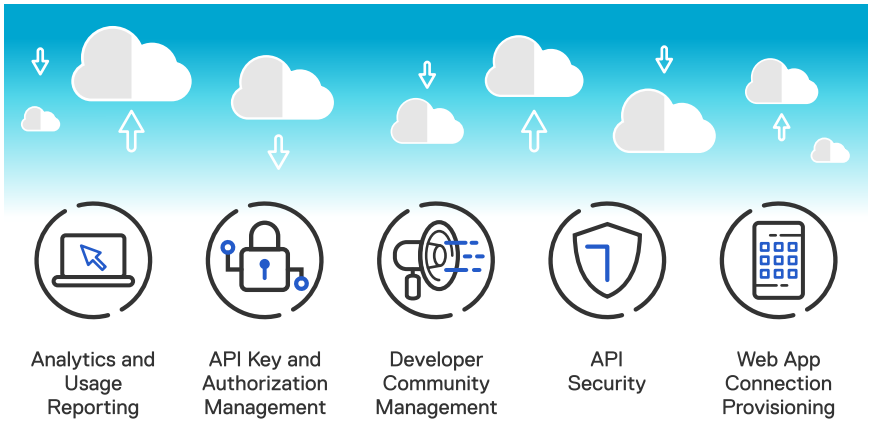


Figure 22. Common uses for API management

The Essence of Input and Output Pipelines

Input and output pipelines are a conceptual framework to help people from different parts of the enterprise value chain understand how their people, applications, services and data should interact to achieve a desired outcome. Three fundamental questions are critical for any desired outcome:

1. *Where* should the data come from, and to *where* should it be staged for processing?
2. *What* does the information tell me after it's been processed?
3. Now that I understand what the data is telling me, *how* do I *orchestrate* my desired outcome?

The answers to these questions provide the model framework for the workflow. Note that the left, input side of the architecture includes raw data that is staged into a persistent, durable store and processed, then written to an output derivative durable buffer where post-process data consumers can orchestrate an outcome.

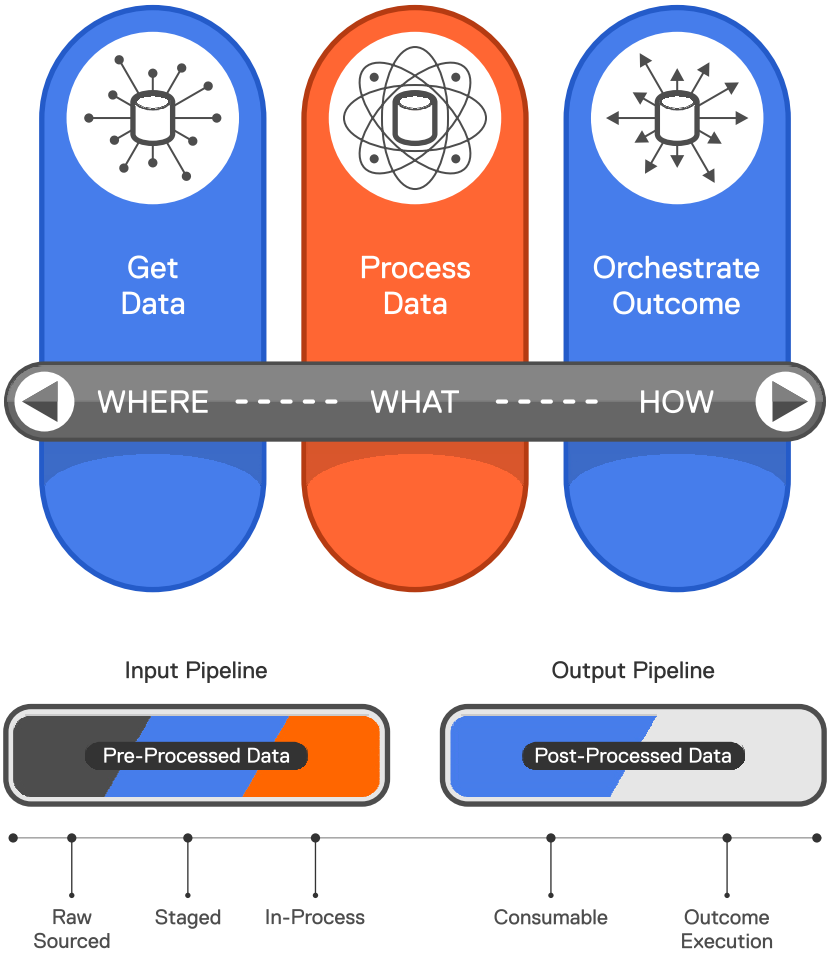


Figure 23. Summary of the input-output data pipeline

Inside the model is where we create the pipelines themselves. But the successful route to achieve this is to *start* with the desired outcome. The clearer the outcome, the simpler the model. The simpler the model, the easier it will be to build the pipeline. After clearly stating the desired outcome, we can uncover the data sources, build the buffers, include the processing engines, and achieve the outcome.

CHAPTER 5

A Dell Technologies Point of View on Data Pipelines

Building modern data pipelines required for a multicloud environment can seem daunting. As we've seen in the previous sections, the philosophy of "*transform or else"* can make you feel that you must do something or face disruption. But building data pipelines doesn't have to be a zero-sum game. While enterprises must modernize to benefit from newer tech to keep up with customer demand and stay a step ahead of the competition, leaping into Estate 2 architecture and abandoning Estate 1 without a clear path is ill-advised.

Maybe some organizations can afford high rates of customer churn commensurate with capricious *estate-hopping*; but the overwhelming majority simply cannot take that risk. While we appreciate the passion to rip and replace legacy systems and processes in the spirit of progress, we do not necessarily recommend it for each organization. Instead we advise a more pragmatic approach. It is best to buttress *some* existing systems with better hardware and upgrade monolithic systems until the safety netting required to transition to an Estate 2 architecture is possible.

In fact, one long-time Dell Technologies customer calls this the 40:40:20 rule[34]; that is, 40 percent of the applications—*new* applications with no associated tech debt—can be built with an Estate 2 framework. Another 40 percent of current applications are deemed *eligible* for transformation, but will require a journey—typically several years—to be converted from an Estate 1 paradigm to Estate 2. And the remaining 20 percent will never be converted. Today, the core of the business either runs on legacy systems or is, in some way, dependent upon them. Until such time that data can be accessed reliably from these legacy systems, or there are no longer dependencies on them, they will be maintained; but not transformed. Fortunately, Dell Technologies can help to expose some of the data from legacy systems so that integration with modern architecture or applications is possible. In other words, for the last 20, we propose that it is possible with some of Dell Technologies' current solutions to make some portion—maybe half—to be eligible for a modernization journey.

34 At the time of publication, we did not receive authorization to disclose the customer name

Whether data is generated from new, “greenfield” systems, or must be pulled from complex but established “brownfield” systems, it is important to start with the outcome. From here, compare the to-be state with the as-is environment, and break down the requirements for where the data must come from, what data must be processed, and how the outcome is to be orchestrated.

From Source of the Input Pipeline

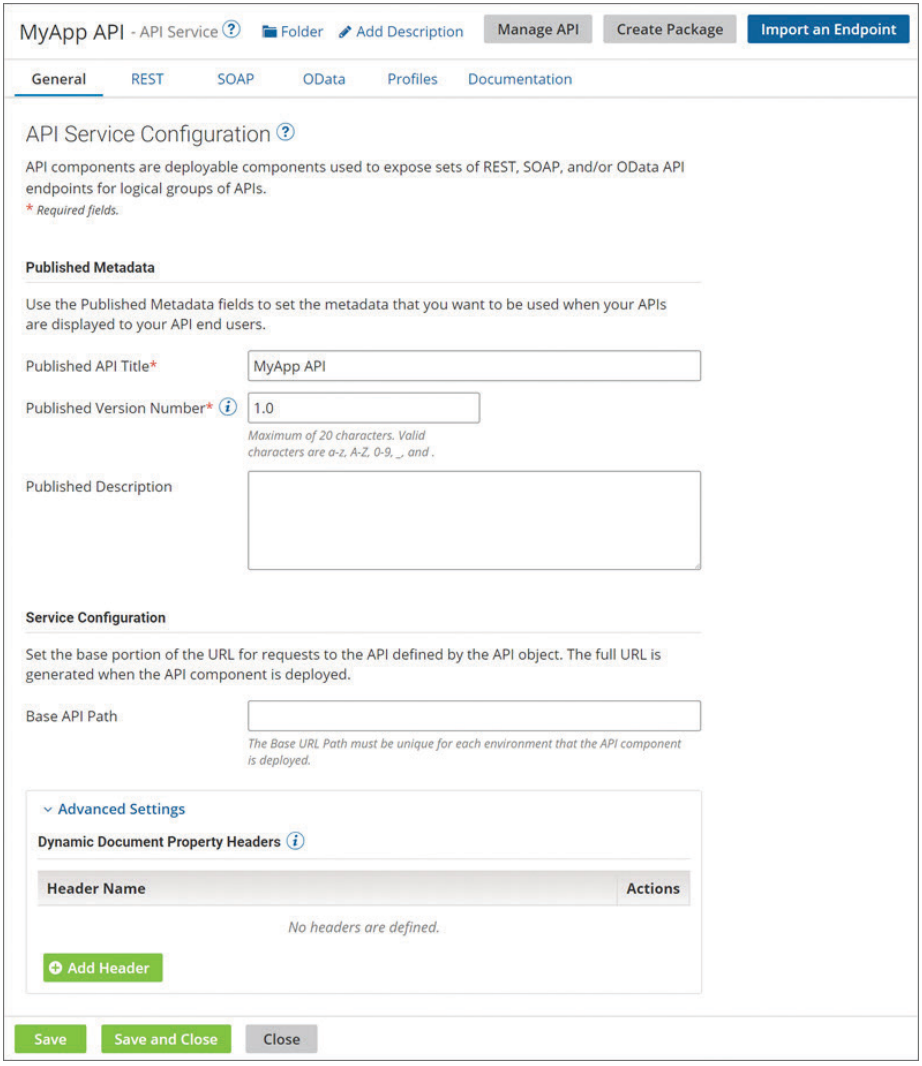


Figure 24. Boomi API Creation interface. (Source: Boomi)

A modern approach to application development means that organizations need to have the tools necessary to design processes to connect SaaS applications, cloud-native applications and on-premises applications. When an organization has the ability to connect to any application within any infrastructure, then digital transformation is within reach. Finding data, then cataloging and preparing the data to create clean sources of truth upon which any process can rely, makes the presentation of data to consumers trivial.

Pipeline data should be easily consumed and produced. Often this is done through an API, but data can also be published or subscribed to as events.

Dell Technologies provides most if not all of the components necessary to make an organization's data available, secure and consumable. In fact, every component for the pragmatic data strategy discussed in Chapter 3 is available right now, including API creation, exposure and management, along with open-source connectors and event driven architectures. From input pipelines to a limitlessly scalable bus to connect endpoints, through outcome orchestration, Dell Technologies can ensure that its customers' data pipelines are modern and transformations are within reach.

In the rest of this chapter, we will look at three additional components that should be at the core of every modern enterprise data fabric: (1) Artificial Intelligence, (2) a multicloud pipeline orchestration platform, and (3) a massively scalable, Kappa event-driven architecture as the data backbone.

Starting first with a massively scalable EDA, **Pravega**[35] can handle the flow of any type of data source, database, batch process or event stream. This storage engine is a software abstraction that allows for the writing to or reading from a persistent data structure. The application-layer interfaces, resource management, and drivers necessary to interact with file systems are handled with very little work. Pravega is an open-source EDA that was developed by Dell Technologies and is one of roughly a dozen EDA partners with which Dell Technologies solutions are compatible.

35 https://pravega.io

Pravega

Traditional ETL pipelines involve having a specialized broker that handles all ingestion needs, durably buffers data, and serves that data in near-real time to streaming applications that are subscribed to it. Generally, events are published to Topics and may be further partitioned. A *separate* ETL pipeline funnels data from this broker into a traditional data-at-rest storage. The architecture looks somewhat along these lines, as shown in Figure 25, below.

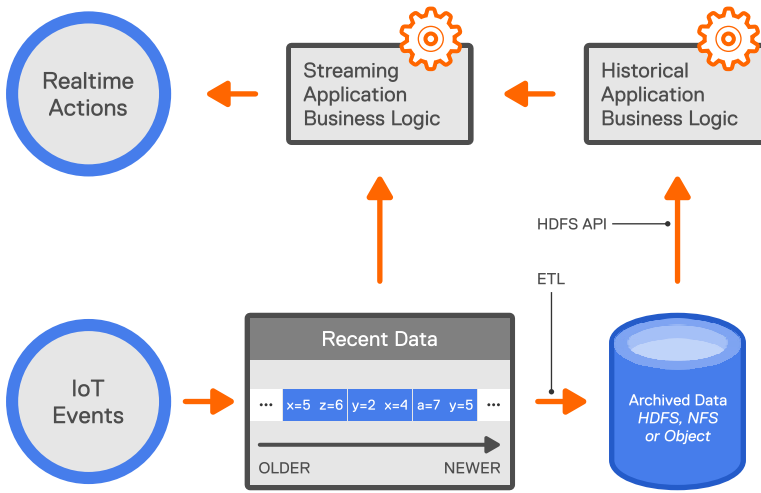


Figure 25. Event Driven Lambda architecture example

While such an architecture serves both streaming and batch (historical) processing needs, it is more complex than it needs to be. One of the biggest challenges is that there is a need for two sets of tools, one for streaming and one for historical data. Due to the mismatch between the APIs and semantics of streaming data vs. historical data (often stored as files or objects), organizations need to write applications that handle both types (and of course, manage the data ecosystems that come along with them, a true operational overload).

A modern ETL pipeline, made possible by using a streaming storage such as Pravega[36], aims to simplify this ecosystem by *unifying the streaming and batch data* into a single API. Figure 26 shows batch abstracted into a Kappa architecture.

36 http://blog.pravega.io/2017/04/09/storage-reimagined-for-a-streaming-world/

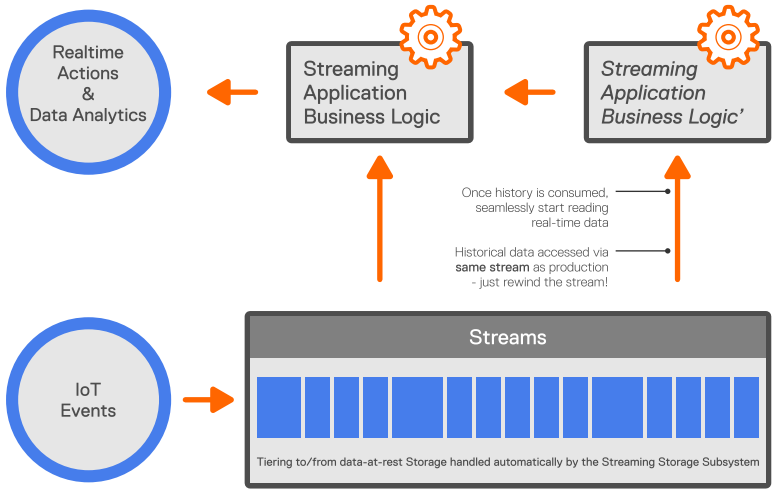


Figure 26. Batch abstracted into a Kappa architecture

While there still exists a *durable buffer* and a *data-at-rest* storage system, the modern streaming storage system will tier data automatically from one to another and, best of all, expose it through a single set of APIs to the user. This tiering work is a transparent process handled internally by the streaming storage system, also referred to as the Stream Store.

However, *data tiering* is not the only feature a modern Stream Store should provide. It is sometimes impossible to predict the ingestion patterns of incoming data. There may be periods where there is little to no activity; and there may be periods where there is so much data to be ingested, that the system is about to fall over. The Stream Store should allow not just horizontal scalability (that is, throw more hardware at it), but also *stream* scalability. This means that even a one-thousand-node cluster will not be of much use if all the ingestion comes into a single stream, and the Stream Store cannot make use of all the hardware available. A *smart* Stream Store auto-scales its streams internally, based on the perceived ingestion load, makes use of more hardware when needed, and relinquishes those resources when the load diminishes.

When we introduced the *durable buffer* earlier in the book, we mentioned some of the problems that these buffers help to solve. One of those was the guarantees that the Stream Store can provide to its users. Developers can simplify their applications greatly if they are made two very important guarantees by the Stream Store: *ordering* and *exactly once delivery*. Events must be presented to consumers in the same order in which they were ingested, and they must be presented exactly once (no duplication or loss).

A modern Streaming Storage System that implements *auto-tiering*, *auto-stream-scaling* and guarantees event *ordering* and *exactly-once delivery* is **Pravega**[37], available free-of-charge via open-source[38]. A Stream Processing pipeline using Pravega can be sketched along the following lines, as shown in Figure 27.

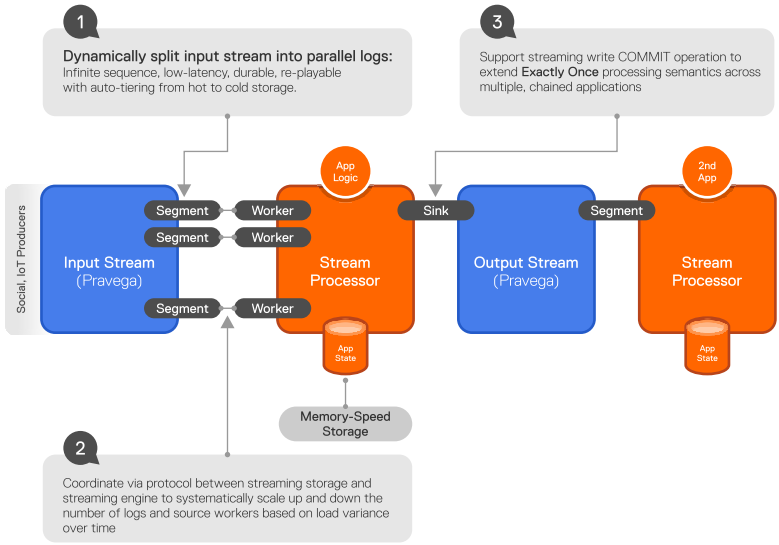


Figure 27. Pravega data pipeline
(Source: https://www.pravega.io)

The Pravega Streaming Storage Subsystems manage the *Input Stream* and the *Output Stream*. Events are ingested into the Input Stream, where *auto-scaling* techniques are used to spread the ingestion load across the cluster (when needed) and release resources when load winds down. A *Stream Processor*, such as Apache Spark[39] or Apache Flink[40] can be used to analyze the Input Stream (both in real time and for historical processing) and write its results into an Output Stream, which can be further used as a source for another Stream Processor. The pipeline can be extended to include as many Streams and Stream Processors as desired by the pipeline developer.

37 https://pravega.io/

38 https://github.com/pravega/pravega

39 https://spark.apache.org/

40 https://flink.apache.org/

Boomi

As we discussed in earlier chapters of the book, the outcome is fundamental to why we need to engineer a data pipeline in the first place. The simple reasons why decisions are made to orchestrate an outcome are to increase something, decrease something, or avoid something. In order to do these things, the data pipeline should be configured to provide the properly processed data to the Output pipeline so that the outcome can be executed and can also be refined.

For many enterprises with which we work, necessity drives the creation of desired outcomes; it often comes from customer demand, and customer demand is articulated from the business to IT. This means that proper orchestration of an outcome includes people collaborating with each other well to understand, design, build, execute and refine the process.

Orchestrating the outcome has been the sweet spot for Boomi since its inception. Boomi has been part of the Dell Technologies family of companies for more than a decade and provides a wealth of services. In 2008, Boomi became the first integration Platform as a Service (iPaaS).[41] In the years since, Boomi has evolved into a full-stack, low code application development platform with intelligent data services.

Want to integrate your ERP system with any other applications, create chatbots, build dashboards, or integrate with just about any endpoint in your data fabric? Would you like to build an entire web application and configure all of the processes to make it work without knowing how to code? Want to get through the backlog of projects in a fraction of the time it normally takes? Boomi can help. From the processed data buffer, upstream applications can be integrated together to form a business process to achieve a desired outcome.

41 https://boomi.com/platform/what-is-ipaas/

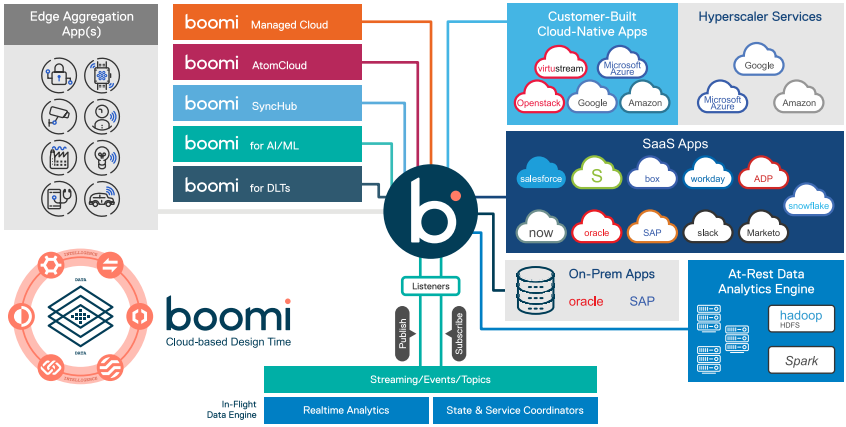


Figure 28. Dell Boomi platform

Outcomes with Boomi are achieved through a low-code graphical user interface, so that the integration or development teams can dramatically reduce the time it takes to integrate applications. It can even empower business to conduct their own self-service integrations. Boomi provides API exposure, presentation and management, even if this means that data are locked in monolithic applications or stored in mainframes. And the Boomi platform does all of this, so that our customers can orchestrate outcomes within Estate 2, and even in Estate 1 frameworks effectively.

Estate 2 Multicloud Pipelines for AI

Staging for ML or DL Processing

Outcomes can be orchestrated after one pass through a pipeline. At other times, getting to a stage where an outcome can be achieved takes multiple iterations of refinement, as is often the case with data science or analysis. For example, raw data may have to be cataloged and prepared as a step before machine learning (ML) or deep learning (DL). If we consider the Input and Output Pipelines, raw data would be harvested from the source of Input Pipeline and processed, before it is dropped off to a derivative buffer of an Output Pipeline. The processed data for a Data Science pipeline may view the processed data as “raw data” within its Input Pipeline source process, shown in Figure 29.

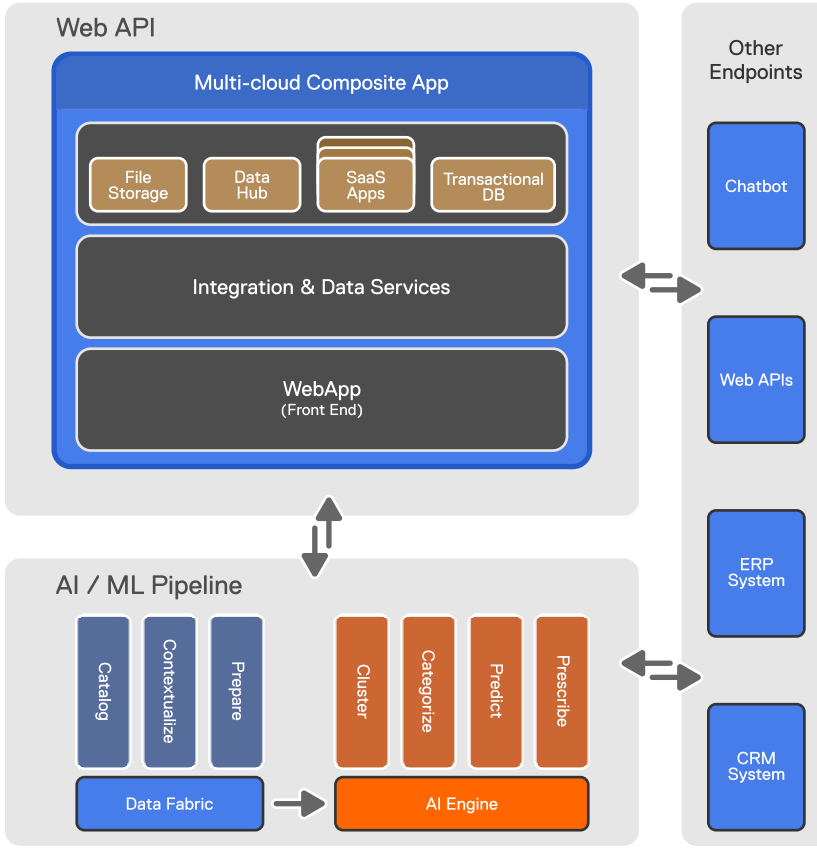


Figure 29. Architectural Stack for Customer 360

Before we can dive into AI, we must put a fine point on everything that we've discussed in this book, up until now. The desired outcomes you obtain are a direct reflection of the quality of the data pipelines that you engineer and the quality of data that you have flowing through them.

If your data pipelines can handle any variety, velocity and volume of data, then you're off to a good start. If your environment can elastically scale compute resources independently of storage, you're doing better. If you have well-established pipelines already and are comfortable with your tech debt, or you have already deployed a Kappa architecture, then you're doing very well. And you are really at the top of your game if your pipelines can listen for state changes in your application endpoints and if activated, can automatically route data to the correct sequence of pipelines so that desired outcomes occur. With these things in place—Kappa architecture (Pravega), a multicloud integration and data management platform (Boomi), and other tools that we

did not discuss—we're ready to take advantage of the benefits of Machine Learning (ML) and Deep Learning (DL).

In addition to the list of "pluses" from the previous paragraph, we suggest that a well-engineered data pipeline is part of an enterprise-wide feedback loop. This feedback loop necessarily is part of a constant flow of dynamically streaming data within a continuous, closed loop system. This means that changes in the state of an application or data should be detectable such that any dependent application, service, function, or relevant human will respond accordingly.

We also suggest that the data is well prepared for any analysis conducted by ML or DL algorithms. In addition to the application byte data that AI engines process, they also can process metadata, messages, and logs. It is important that each of these data sources is properly encoded (labeled), free of errors, accurately schematized, correctly formatted, etc. Additionally, it is likely that data may contain personally identifiable information (PII—credit card info, SSN, PIN, Name, Addr, etc.). Filtering data to remove PII is no easy task, but it is a necessary step for further processing. Fortunately, Dell Technologies can help, with Boomi Data Catalog & Prep (DCP) and Dell Technologies DataIQ. Being able to discover data sources, catalog them and, lastly, prepare them for your data pipeline is exactly the purpose of these tools. This way, you can get the most out of your pipelines. With clean data, you're ready for maximizing the value of your data throughout your entire data fabric.

Prior to the use of AI—and still prolific today—humans have employed conditional logic in some way to expose and handle problems (for example, if-then statements such as, *if: temp > 50, machine.stop()*). Exposing problems is often deterministic, meaning that a human is trying to apply rules to anticipate certain conditions. For example, a bug in code can be ruled out methodically via some binary tree of probable causes. Determining the root cause of an issue when starting with two, three or four variables is relatively simple; but what about 20, 30, or more? How can you correlate features? How do you classify them? How do you figure out patterns in data that *seem* dissimilar? How do you predict a change in the future? How can you prescribe a recommendation? What if you don't even know if you have the *right* data to get any worthwhile answers? What if a non-deterministic approach is needed to handle conditions? For all of these reasons and many more, ML and DL are necessary.

When AI is used in a scalable data pipeline that supports a continuous feedback loop, then greater efficiencies in systems that support the enterprise can be achieved, resulting in better outcomes. Anomalies can be detected within the state of an application. Future outcomes at a micro- or macro-level can be predicted. Recommendations to humans can be prescribed. New limits can be applied. Even non-deterministic rules can be applied to conditions within a pipeline.

The desired outcome for the user story is to minimize resolution time for a given customer issue. And this architecture shows how a composite application made up of multiple endpoints coupled with a data management pipeline, integration platform, data cleansing and, ultimately, a prediction engine can optimize a workflow to minimize downtime. We are not going to delve into any specific use case in this book. Instead, we shall discuss some different ways that artificial intelligence (AI) can help us achieve outcomes. In the rest of the book, we will focus on how to best reach desired outcomes with different applications of Machine Learning (ML) and Deep Learning (DL).

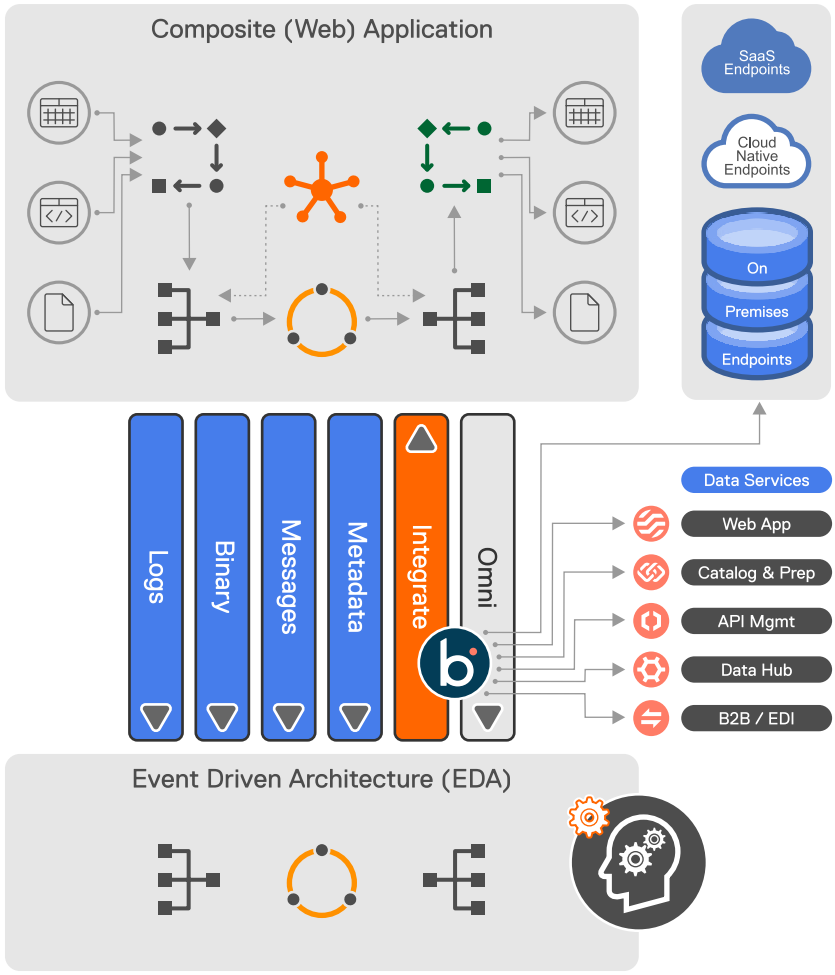


Figure 30. Continuous Workflow for Input and Output Pipelines.

Machine and Deep Learning algorithms can take advantage of this continuous data pipeline. Because Dell Technologies' Streaming Data Platform is able to handle batch, event-based and real-time workflows, machine learning and deep learning engines can offer real-time capabilities to applications where it was very difficult, if not impossible to do so before Pravega's approach to kappa architecture, and Dell Technologies' approach to data fabrics.

Data for Artificial Intelligence

From the data fabric, Input and Output pipelines take advantage of the necessary endpoint connections across clouds and data services to make artificial intelligence work. Data is at the heart of training an AI model. And good data orchestrated through solid pipelines means that AI can enable significant business outcomes.

The importance of artificial intelligence cannot be overstated if organizations want to exist in the economy of the future. To have a strategy of growth or even survival, AI—machine learning and deep learning—must be at the heart of organizational processes. And just like a healthy heart needs oxygen and good blood flow, so too is a steady stream of clean and accurate data important to an AI engine.

In this section, we show how you can apply the data architectural patterns we have described to your implementation of AI. Please note that we will use terms like Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) interchangeably.

For those new to AI, it can seem like magic. After all, AI can perform computations with orders of magnitude more data in a fraction of the time that it would take manually designed processing steps. Think of some real world examples that we use every day that were the stuff of science fiction just a few years ago:

- Hailing a freelance driver via ride-share app
- Asking your home personal assistant device to order and deliver groceries to your home
- Recommending clothing based on what you've read earlier in the day
- Being able to detect if you've contracted a virus, like COVID-19, through the *way* that you cough[42]

These are amazing examples; but are just a few of the myriad AI applications that are present in your everyday life.

42 https://news.mit.edu/2020/covid-19-cough-cellphone-detection-1029

AI is undergoing rapid changes and adoption and is fueling a new kind of world and a new economy with it. Iansiti and Lakhani propose in their book *Competing in the Age of AI*, that only organizations that can transform into data and AI companies will be able to survive in a future economy:

"AI is becoming the universal engine of execution. As digital technology increasingly shapes 'all of what we do' and enables a rapidly growing number of tasks and processes, AI is becoming the new operational foundation of business—the core of the company's operating model, defining how the company drives the execution of tasks. AI is not only displacing human activity, it is changing the very concept of the firm…transforming the nature of organizations and the ways they shape the world around us."[43]

Thinking of AI as an execution engine, and not magic, is a key insight. AI, ML, or DL, is really just a powerful collection of physical processors (CPUs, GPUs, TPUs, IPUs, FPGAs, etc.) with abstracted software—algorithms—designed to orchestrate these processors in a way that powerful tasks can be performed. When data is fed into an AI engine, these tasks can take minutes or seconds to perform versus the weeks, months or, years for humans to complete.

If your organization is to compete in the economy of the future, then AI should be at the core of each process. Organizational direction to determine the appropriate outcomes for the enterprise is precisely in the domain of the architects that design the fabrics from which the pipelines are constructed. The business analyzes data relevant to their own context. This allows for Enterprise Architects to transfer the business value and the user story into a technical framework that can be further refined into technical specifications. The Enterprise Architects provide the blueprint to the Solution Architecture team so that they can choose the general tool chains that will be used to implement, test, and refine the data pipelines and the AI engines appropriate to achieve the desired outcome. From here, technical architects and data engineers can use specific endpoints, tools and pipelines that will convert the technical specifications into the actual workflows that emerge from the appropriate data, pipelines and AI engines.

The AI engines and the data sources that feed them are used to achieve desired outcomes. These desired outcomes must be able to draw on any kind of data from any relevant source regardless of its volume, variety, or velocity. From each endpoint source to each endpoint target, data should be refined and staged to present to the AI, such that the data can be processed. Once the AI processes the data, the output is ready for consumption. It is from here—from these modern enterprise data pipelines—that your organization can orchestrate its desired outcomes.

43 Iansiti & Lakhani (p.3)

From Data Pipelines to AI Engines and Back Again

Data is the raw material used for machine learning. The data fabric acts as the resource layer to supply the pipelines that feed the AI engine. The data pipelines that feed a given AI engine form a composite AI pipeline. For clarity let's distinguish between the AI *engine* and AI *pipeline*. Within the context of an Estate 2, multicloud environment, an AI *engine* denotes the actual machine learning or deep learning processes of model development (we'll discuss this soon). The AI *pipeline* is a series of components—endpoints or services—that constitute a sequence of functions between the components. Within a multicloud estate, data of any variety could be sourced from anywhere.

There is a lot going on in the AI engine, and for each major, or *parent,* stage of a given AI pipeline, there are child-processes that branch data into derivative pipelines of their own for refinement. Child-processes may merge and split multiple times before data is ready to transfer from one parent process to the next. All of this, of course, is made up by one or more—likely *many*—pipelines tasked for preparing data before they are ready for model training.

By *model*, we mean something different than a *data, architectural or organizational* model. We mean a *machine learning* or a *deep learning* model. Joel Grus refers to a model as a specification of a mathematical or probabilistic relationship that exists between different variables.[44] In other words, a model looks for patterns in data. If there is a pattern that lurks within one dataset, perhaps that pattern can be applied to others; therein lies the value. AI functions predict, prescribe and classify. For AI to be effective, it needs accurate data from which to learn. To have accurate data, good pipelines must be implemented. The better the data the model can train on, the better the model will be able to perform when it sees new information that it hasn't trained on.

For each stage within an AI pipeline, there are sub-steps. For any of these sub-steps, data may be collected from many sources. Data *collection* can mean the physical copying (replica of the original) or migrating (movement of original) of the source of data. This is the case for having a repository of structured and unstructured data, like a data lake. Additionally, data collection refers to assembling the *metadata* of the original data.

Endpoints within AI pipelines rarely need to migrate or copy data from one stage to the next. Doing so would be time and memory intensive. Instead, a more compact representation of the dataset enables efficient processing between the pipeline components. This efficiency makes metadata a key element to the communication between endpoints in a pipeline.

The metadata also becomes a powerful way to prepare data so that it can be used for the AI model. Once the data has been collected, its metadata needs to be stored and cataloged so that proper refinement of the data for

44 Grus, p.153

the model can commence. Here, raw data is stepped through a sequence of transformations so that the algorithms that use the data can figure out which model to apply to the dataset(s) being examined.

AI algorithms train models quickly and accurately by mapping, storing and retrieving metadata about the full dataset by *vectorizing* the features (such as keys, column headers, and so forth) of the dataset. Vectors are mathematical representations of objects, and vectorization is the process to turn these objects and their features into a set of numbers. Through vectorizing the data, space in memory and processing time are saved.

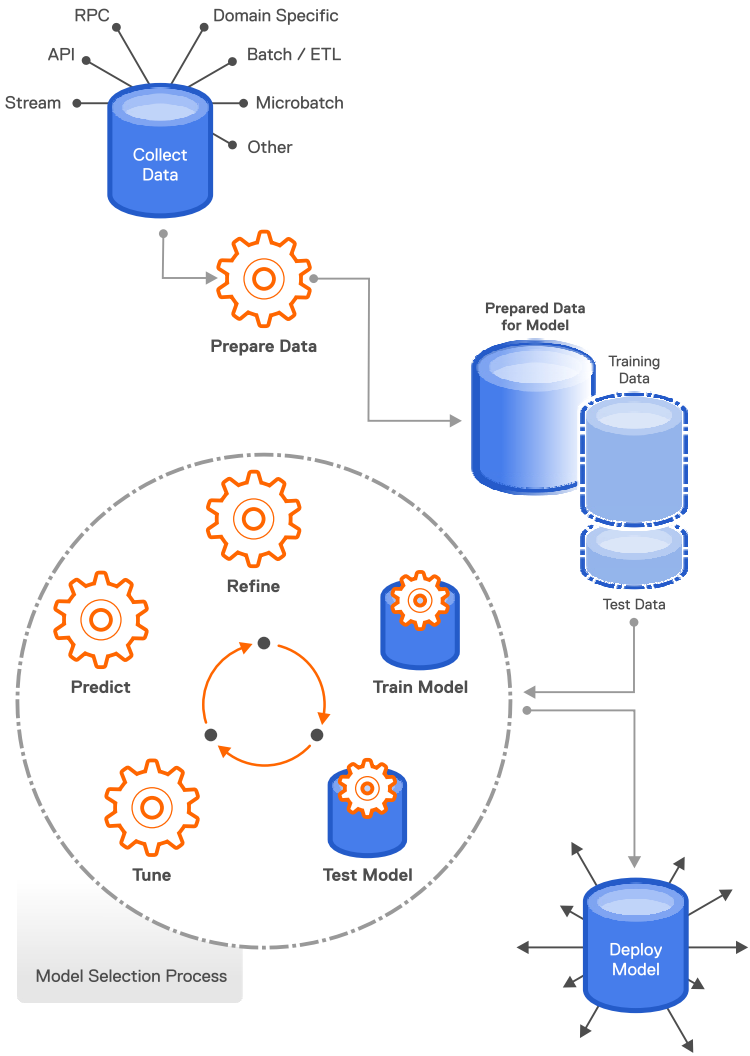


Figure 31. AI Pipeline

When a model is chosen, an iterative process is applied to the dataset(s) in question until the mathematical relationship between a dataset and the model that describes it is sufficient. A model's sufficiency is often judged by the frequency of its correct prediction.

For example, if a company's trained model determines that customers ages 35–54 are 53% more likely to buy products from its eCommerce site when the website background is in "dark mode," and that model is 91% accurate, it would probably be a good idea to change the background of the website. After all, doing so, could mean ~48% more sales. But if 35 to 54 year-olds are not the target demographic for the company's products, changing the background may be a waste of time, or worse. Alternatively, if 55- to 74-year-olds *are* the target demographic, and the same model predicts a 28% sales *decrease* when the website is in dark mode, then leaving the light background as default may be the better choice.

Choosing a model involves a repeating process of training, testing, tuning, predicting, and refining until optimal accuracy relative to a goal is reached. We'll summarize the steps quickly, here; but there are great resources, including some listed in the bibliography, if you want to learn more about how to build AI pipelines or develop AI models.[45]

Although Figure 31 greatly simplifies what is involved with choosing a model, the steps are relatively simple, especially since there are so many tools available to adjust data and AI models. To train the model, use functions within your favorite AI platform to split up the vectorized dataset. A large percentage of the full dataset should be used for training the model, and the rest for *testing* (a typical train-to-test data ratio is 80:20). Then determine what features need to be dropped from, added to or modulated in the model. This step is called *tuning.*

Please note that this book does not address the data science or procedural steps that would be required in any implementation, such as statistical inference or controlling for bias; our focus is limited to the foundational concepts of data pipelines.

Once the precision of the model is tuned to a desired point, then assessing the accuracy of the prediction when the training and test data is shuffled comes next. This "shuffling" process will occur a number of times (called "epochs") until the accuracy is sufficient. If the accuracy is deemed to be good for deployment, the model can be deployed to the processing layer of the application, and the application can be exposed as an API or can be accessed through a remote procedural call. Either way the AI can now become part of the data fabric and can be incorporated into one or more data pipelines.

45 There are many books, articles, videos and Massively Open Online Courses (MOOCs) to dive into these details. Of course, for those interested in making a career of developing AI models, Data Science is all the rage and universities are more than happy to teach you about one of the hottest jobs around. You can also talk to our AI experts at Dell Technologies.

If the model is insufficient, refactoring is required. Refactoring could simply involve retuning some of the parameters of the current vectorized dataset, and re-train, or it could mean going through the prep process with all of the data. If the data is not adequate, refactoring could also involve starting over from the very beginning and connecting to new data sources. Regardless, at some point either the model or the data will need to be adjusted, even if they were once good.

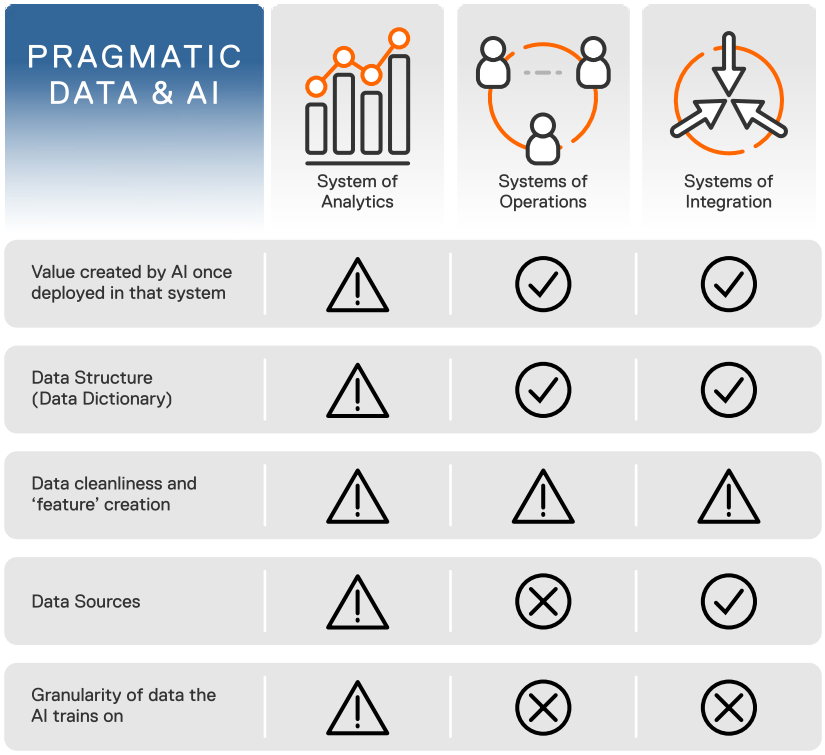


Figure 32. Pragmatic Data and Pragmatic uses of AI (adapted with permission from Aible AI)

The painful reality of AI that there isn't (yet) an Easy button for its use. Models take time to develop. And in order to develop these models, it helps to know exactly what kind of outcome you want. As we discussed in Chapter 2, placing an Estate 1 architecture into the cloud and thinking that it will simply be more efficient as a result doesn't work. Similarly, incorporating an AI model into an existing process just for the sake of "having AI" is also likely to fail. Remember: *outcome, first.*

There is good news, though! Not only do modern enterprise data pipelines greatly facilitate the deployment of AI, Edge and other emerging technologies in your multicloud estate, they can also facilitate the use of best of breed SaaS endpoints in the cloud. Rather than building your own AI pipelines from scratch, you can elect to offload the AI computation to SaaS services that you buy. This means that the responsibility of your pipelines can be to reliably input source data to and the SaaS AI engine, and get the results from that engine to feed the dependent applications for the processed data. Many customers toil over the value of building versus buying their own AI pipelines.

Figure 32 encapsulates simple dimensions of value that may be useful for your journey. According to our AI SaaS partner, Aible, and Dell Technologies company, Boomi, there are three, essential systems to think about when deploying AI: *analytics, operations* and *integration.* In essence, systems of *operations* are the actual infrastructure managed by IT. Systems of *integration* are also in scope with IT; but these systems are focused on connecting disparate systems and managing data flows between them, as we've discussed. And finally, systems of *analytics*, which are designed to provide valuable insights to the organization.

Within each of these systems are five general dimensions to consider when trying to realize the value that AI can bring to your organization. What emerges is a matrix that can act as a scorecard of sorts, with the idea that the more checkmarks you have, the better the likely outcome will be.

A *System of Analytics* is essentially a series of one or more pipelines (as in Figure 30) that trains on data to build a model that represents the ostensible reality into which clean data will flow. The System of Analytics is tuned typically for the best possible model. For an optimal model, well-cleaned data is typically fed into this AI pipeline. The problem with this approach is that the world is messy, and clean data, while good for the model, may be inadequate for the real world. This can lead to either over- or underfitting. For example, if you are looking to determine the probability for a sale to occur in the fewest number of visits and you are training against a model that is highly accurate on metrics that aren't important to a sale, then you will likely get bad predictions from the AI model.

A *System of Operations* means that your AI model can be trained against data that is reflective of operations as they are today. These systems are great, especially at first, because they train on data that is real. The model may not be perfect, but at least it's indicative of the right landscape as it exists at the time when the model is deployed initially. The problem here, though, is the model's adaptability to changes over time. If it is difficult or time-consuming to re-train and re-deploy models for whatever reason (typically over new data sources in a timely manner), then the predictions from an operational model may be insufficient. After all, this model is not designed to be flexible once deployed. The question to ask yourself is this: *how easy is it for my production model to take on new data sources once deployed?* If the answer is *not easy*,

then you should think hard about deploying this model into production. After all, the world constantly changes, even if your model does not.

A System of Integration means that an AI model can incorporate new data sources relatively easily; however, the model may not be easy to tune to the new input pipeline. Retraining AI algorithms typically takes time. As with Systems of Operations, Integration still requires model updates. This means that if the model can take in new data sources; but the model does not adjust appropriately to the newly integrated data, then predictions may lead to inaccurate predictions.

Having an approach to optimize Systems of Analytics, Operations and Integration is key to building your own AI engines or simply supplying data to AI services. The important point to keep in mind is that for AI to be valuable, it is essential that *each* system work in concert. Perhaps you will not be surprised to know that while there is no clear winner shown above, Dell Technologies and ecosystem partners, like Aible, do allow users to achieve a complete list of checkboxes. With the tools that allow our customers to source data, catalog it, prepare it and be able to incorporate new sources of granular data to train the AI engine, all three systems can be satisfied. And this means that you can quickly and efficiently derive value.

We hope that you find that data pipelines and the fabric from which they are made are simple concepts to understand, easy to architect or engineer, and quick to deploy. With the foundations presented here, you should be equipped to learn more and to ask important questions of your staff, peers and leadership to enable your organization to thrive in the economy of the future.

CHAPTER 6

CONCLUSION

To transform, an organization needs to embrace change. The past several decades of rapid digital growth are mere prologue to the changes already unfolding. Some predict that the years between now and 2030 will see more technological advancement than in the sum of all previous decades in human history. One thing is certain: the 2030 digital landscape will look much different from the landscape of today. Transformation is needed to compete in the economy of the future. To transform, harnessing data is essential. The time to understand and build modern enterprise data pipelines is now.

Enterprises and organizations need to make sense of customers' buying (or selling) patterns and behavior, as well as external factors that may influence either. HR must hire the best talent available. Supply or value chains need to be optimized for exact demand. Leaders must assess when and how to enter or exit markets. These are all difficult things to accomplish, but organizations that do them well are likely to be the market leaders of tomorrow. If they're anything like Dell Technologies, or the other top 19 global companies who have invested in digital transformations,[46] they will need to invest in technology to engage and retain the best people.

The best people become the catalyst for digital transformation. With them come new ways of solving old problems, and new platforms are needed to solve greater challenges. These new platforms were the focus of this book. Both the 3-tier infrastructure with monolithic application-building approach of Estate 1 and the multicloud, microservice philosophy of Estate 2 architecture have a place in the modern data center. Because of the tooling and pipelines that are now available to accommodate application and data services in any cloud, developers can create and deploy new ways to harvest data faster than before. Exponential data growth requires tools that can unlock data wherever it is located and harness its value.

We hope that Estate 2 architecture, input and output pipeline and Kappa architectural concepts can help you understand how you can create modern

46 https://hbr.org/2019/09/the-top-20-business-transformations-of-the-last-decade

data pipelines within your organization, regardless of cloud. Moving on-premises applications and data entirely to the cloud is not simple. Stakeholders want their applications and data services to be simple to use, feature-rich, and as resilient as possible. They want in modern applications and orchestrated outcomes that distinguish them from their competitors. And they want to provide the best possible outcomes and experiences for their consumers.

Transformation does not simply mean migrating an application to the cloud from an on-premises environment. The right tools and resources are necessary for a viable multicloud architecture to keep up with the pace of market changes without abandoning current technical investments. Creating modern multicloud data pipelines is at the heart of any transformation, and the transformation starts with knowing the desired outcome.

This book sought to describe the architecture of the data pipelines within a multicloud enterprise application and data environment. Conceptually, input and output pipelines are very similar to traditional infrastructure of the past. Source data lands on storage (represented by the blue cylinders in Figure 31), red gears resemble compute and volatile memory, and post-processed data takes residence in a derivative storage environment. All the source or raw data input is staged on a durable buffer. From the buffer the processing engines of one or more applications refine the data in accordance with the desired objective. To the right of the processed data buffer is a sequence of processed data that are merged into a data stream to achieve a desired outcome. It is that simple.

The fundamental reason that we have modern data pipelines in the first place is to orchestrate desired outcomes. How do you do this today, and what processes do you have that support each outcome? For a given process, where is the data coming from? When you process the data, what is it telling you? How are you correlating all your data, metadata, and logs? Are you using AI? Are your apps able to signal others to trigger new streams? How can you refine your data streams or harvest data from new sources? These are just some of the questions you can ask us. We are here to help.

To sum up, these are the three main points we hope you take with you:

- **The changes in thinking that are necessary for true transformation.** Transforming an enterprise's data pipelines begins with a change of mindset, culturally and architecturally. Unlike legacy 3-tier approaches to information systems of the past, the shift to a multicloud strategy requires thinking *small* about applications or services while thinking *big* about data. Chapter 2 focused on these topics by introducing and discussing the organizational concepts of Estate 1 and Estate 2.
- **The vocabulary, architectural foundations, and conceptual framework required for a technical transition.** Chapters 3 and 4 examined architecture patterns that ingest raw data from anywhere, process data through sequenced steps, and stage processed data to the consumer.

We examined Lambda and Kappa architectures, input and output pipelines, and some important vocabulary to help you relate contemporary terms to a mental model that may be more familiar.

- **An understanding of how to apply data pipelines to transformations**. Transformations start with the organization's stakeholders and their desired outcomes, and this leads into how the data of the organization is consumed. The technology required to orchestrate desired outcomes is available. Having the right partner to work with you on this journey is essential. As a company that gives its customers the knowledge and tools necessary to modernize their estate now, while preparing them for changes to come, Dell Technologies can be that partner.

What Can You Do Now?

Begin by articulating the outcomes that you want to achieve, then define and model the processes to achieve those outcomes. Understand the data that needs to be processed in the data pipeline. With the right conceptual framework and understanding of key modern pipeline concepts, this is easier than ever; even in a multicloud environment.

Dell Technologies can assist you with a variety of tools, platforms, solutions and services. Contact your Dell account team about how you can create transformative data fabrics with resilient, extensible and elastic pipelines. Join us at an upcoming event. Invite us in for workshops with your integration teams, application development, development and data operations teams, and your business owners. Subscribe to our blogs and newsfeeds. Download and try our software.

GLOSSARY

Artificial Intelligence (AI): Any device or method that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.[1] Colloquially, the term "artificial intelligence" is often used to describe machines (or computers) that mimic cognitive functions that humans associate with the human mind, such as learning and problem solving.[47]

Batch Process: A computation that takes some fixed (and usually large) set of data as input and produces some other data as output, without modifying the input. [Kleppmann, 2017]

Broker: Computer program that translates the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Used as the intermediary between messages in a pub-sub pattern.

Business Intelligence (BI): The strategies and technologies used by enterprises for the data analysis of business information.[1] BI technologies provide historical, current and predictive views of business operations. Common functions of business intelligence technologies include reporting, online analytical processing, analytics, data mining, process mining, complex event processing, business performance management, benchmarking, text mining, predictive analytics and prescriptive analytics.[48]

Change Data Capture (CDC): Set of software design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data. CDC is also an approach to data integration that is based on the identification, capture and delivery of the changes made to enterprise data sources. CDC solutions occur most often in data warehouse environments, where capturing and preserving the state of data across time is one of the core functions, but CDC can be used in any database or data repository system.[49]

47 https://en.wikipedia.org/wiki/Artificial_intelligence

48 https://en.wikipedia.org/wiki/Business_intelligence

49 https://en.wikipedia.org/wiki/Change_data_capture

Data fabric: An architectural concept for an organization's comprehensive inventory of available (multicloud) endpoints, services and processes that can be used in any combination or permutation to form one or more data pipelines.

Data pipeline: Abstracted software representation of physical compute and long- and short-term memory systems. While data pipelines access physical-layer infrastructure at some point, the type of architecture that is discussed in this book involves pipelines close to the application. Specifically, we discuss moving raw data *into* the pipelines, processing the data, and handling the processed data coming *out* of the pipelines.

Database Management System (DBMS): Software system that enables users to define, create, maintain and control access to a database.[50]

Deep Learning (DL): Part of a broader family of machine learning methods based on artificial neural networks. Learning can be supervised, semi-supervised or unsupervised.[51]

Dequeue: Process where data are removed from the front of or off a queue in sequence.

Durable Buffer: Term to describe an abstracted representation of a persistent data store, in memory or on disk.

Endpoint: Entry point to a service, process or a queue, buffer, topic or stream destination.

Enqueue: Process where data is placed into the back or onto a queue in sequence.

Enterprise Service Bus (ESB): Communication system between mutually interacting software applications in a service-oriented architecture (SOA). It represents a software architecture for distributed computing and is a special variant of the more general client/server model, wherein any application may behave as server or client.[52]

Estate 1: A legacy platform of technologies that is used as a base upon which other applications, processes, or technologies are developed. A legacy platform consists of silo-based infrastructure domains supporting legacy persistent application stacks. These environments may include both physical and virtualized hardware; however, the underlying architecture still follows an amalgamation of decisions made over the past 30 or more years. Characteristics include, but are not limited to: (1) workload pinned to physical blades, (2) traditional HA/DR/Failover/Failback where resiliency is at a hardware level, (3) SAN with traditional storage arrays, (4) multiple levels of ACL groups or DMZs, (5) physical load-balancers and firewalls,

50 https://en.wikipedia.org/wiki/Database#Database_management_system

51 https://en.wikipedia.org/wiki/Deep_learning

52 https://en.wikipedia.org/wiki/Enterprise_service_bus

(6) isolated network domains, and (7) fixed resource allocation. Thus, application programs written for one platform would not work on a different platform. Examples are persistent workload images and CapEx run models. The existence of server, storage or network virtualization does not by itself equate to a software-defined data center (SDDC).

Estate 2: A platform of fully enabled, multicloud-ready, SDDC-supported native cloud application workload frameworks, like PAS and PKS. Characteristics include, but are not limited to: (1) cross data center/cross cloud dynamic/flexible resource pools, workload domains, elastic scalability, and availability zones, (2) network function virtualization, (3) micro-segmentation, (4) geo-fencing, (5) secure tenant isolation, (6) resiliency via availability zones not hardware-dependent, (7) any-to-many dynamic API connectivity between any PaaS, SaaS, FaaS, BPaaS, etc. (8) dynamic/variable resource allocation, sizing and utilization, (9) non-persistent workload images; for example, serverless; OpEx run model; etc. Estate 2 is always a net-new, fully-enabled hardware/software architecture built to support net-new workloads or services, inclusive of proper CI/CD/DevOps-focused resource automation, orchestration, life cycle management, and brokerage. Workloads and the framework can run on different platforms.

ETL extract-transform-load: Process by which data is either copied or migrated from a source database to a target database so that it is suitable for more interrogation, manipulation and queries

Input Pipeline Architecture: Logical framework that includes where the data and data sources are located and how to enqueue them into the processing data pipeline for processing. Once the processing completes, the framework addresses what sense can be made from the processed data. The Input pipeline includes part of the recurring reciprocal data flow between a persistent data buffer holding pre-processed data, an analytical processing compute resource, and a target persistent buffer holding final processed data. This forms part of the "Blue Pipe–Red Pipe–Blue Pipe" (BRB) flow, where there can be *n* blue pipes and *m* red pipes; but there will always be a pre-process buffer and a post-process buffer. Note that the final blue pipe is part of the output pipeline architecture.

Integration Platform as a Service (iPaaS): Cloud-based integration that is a form of systems integration business delivered as a cloud computing service that addresses data, process, service-oriented architecture (SOA) and application integration.[53]

53 https://en.wikipedia.org/wiki/Cloud-based_integration#cite_note-GartnerReferenceModel-1

Output Pipeline Architecture: Logical framework at the point of dequeuing from a post-process buffer where endpoint coordination creates one or more processes to achieve a desired outcome. The output pipeline includes part of the recurring reciprocal data flow between a persistent data buffer holding pre-processed data, an analytical processing compute resource, and a target persistent buffer holding final processed data. This forms part of the "Blue Pipe–Red Pipe–Blue Pipe" (BRB) flow, where there can be n blue pipes and m red pipes; but there will always be a pre-process buffer and a post-process buffer. Note that the final blue pipe is part of the output pipeline architecture.

Machine Learning: The scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data," in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.[54]

Master Data Management (MDM): Method used to define and manage the critical data of an organization to provide, with data integration, a single point of reference.[1] The data that is mastered may include reference data (the set of permissible values) and the analytical data that supports decision making.[55]

Schema: A description of the structure of some data, including its fields and datatypes.

Service-Oriented Architecture (SOA): Software design pattern where services are exposed between applications and accessed by a network.

Stream: Durable, elastic, append-only, unbounded sequence of bytes that has good performance and strong consistency.

Stream Process: A continually running computation that consumes a never-ending stream of events as input and derives some output from it. (Kleppmann, 2017)

Stream Store: A transparent tiering process handled internally by the streaming storage engine.

Storage Engine: Also called *Database Engine*: The underlying software component used by a database management system to create, read, update or delete data from a database.

54 https://en.wikipedia.org/wiki/Machine_learning

55 https://en.wikipedia.org/wiki/Master_data_management

Bibliography

Aragues, Anthony, *Visualizing Streaming Data*. O'Reilly, 2018.

Data Management Association International, *Data Management Body of Knowledge*. Technics Publications, 2017.

Grus, Joel, *Data Science from Scratch-First Principles with Python, 2 ed.* O'Reilly, 2019.

Hapke, Hannes and Catherine Nelson, *Building Machine Learning Pipelines.* O'Reilly, 2020.

Hohpe, Gregor and Bobby Woolf, *Enterprise Integration Patterns*, Addison Wesley, 2004.

Iansiti, Marco and Karim Lakhani, *Competing in the Age of AI.* Harvard Business Review Press, 2020

John, Tomcy and Pankaj Misra, *Data Lake for Enterprises*.
Packt Publishing, 2017.

Kleppmann, Martin, *Designing Data Intensive Applications*.
O'Reilly Media, 2017.

Malaska, Ted & Jonathan Seidman, *Foundations for Architecting Data Solutions*. O'Reilly, 2018.

Özsu, M.T, P. Valduriez, *Principles of Distributed Database Systems*. Prentice Hall, 1999.

Pomerantz, Jeffrey, *Metadata*. MIT Press. Cambridge, 2015.

Volkhover, Anatoly, *Become an Awesome Software Architect: Book 1 — Foundation* 2019. Publisher unknown, 2019. ISBN: 978-1-69727-106-5

MODERN ENTERPRISE DATA PIPELINES

Mike Bachman is the Chief Architect in the Office of the CTO at Boomi. Bachman consults with the Dell Technologies Enterprise Architecture Group, which is responsible for Dell Technologies' largest global enterprises, and works with their Product and Engineering teams as well. He has over 20 years of professional experience in technology consulting, infrastructure architecture, application integration, performance analysis, data strategy, and emerging technology. Bachman is now focused on how emerging technology can solve global-scale sustainability issues.

Haji Aref has over twenty-nine years of experience with data, development, advanced data analytics, ethical hacking, and customer support. For the past fourteen years, Haji has served as chief global architect for innovation across Dell Technology companies, with a focus on data, security, advanced analytics, multicloud, data mobility, big data, AI/ML, and IT resource management. Haji has received numerous certifications, including Level Three Open Group Master IT Architect, and is the holder of many patents and author of numerous publications.

Rick Lemelin, a certified Distinguished Enterprise Architect, has worked in Information Technology for 31 years, across all facets of business/technology optimization and transformation. Currently, Rick leads the Dell Technology Select Architecture Strategy and the Enterprise Architecture CoE. Rick is driven to understand and teach the divergence between legacy and digital business models, including the requirements of next generation application/data behavioral patterns, infrastructure topology deployment patterns, integration patterns, data sharing patterns, and resource consumption patterns.

Andrei Paduroiu is a Distinguished Member of Technical Staff within the Unstructured Data Storage group at Dell. Author of several US patents in the event streaming space, he is one of the founding members of open-source project Pravega, a revolutionary streaming storage system that aims to unify stream and batch processing. Andrei earned a Master's degree from Northeastern University, concentrating on machine learning and information retrieval, and a Bachelor's degree from Worcester Polytechnic Institute.

ISBN 978-173736230-2

9 781737 362302