elastic | The Search AI Company

The Elastic Search AI Platform as a Distributed Data Mesh

Why Do We Collect Data?
It's simple—we collect data to use it. To make good decisions, you need all the facts.

- Some data needs immediate analysis for real-time decisions.
- Some gets stored for later—ideally in the cheapest storage possible.
- But when the time comes, you need to **retrieve, connect, and act on it—fast.**

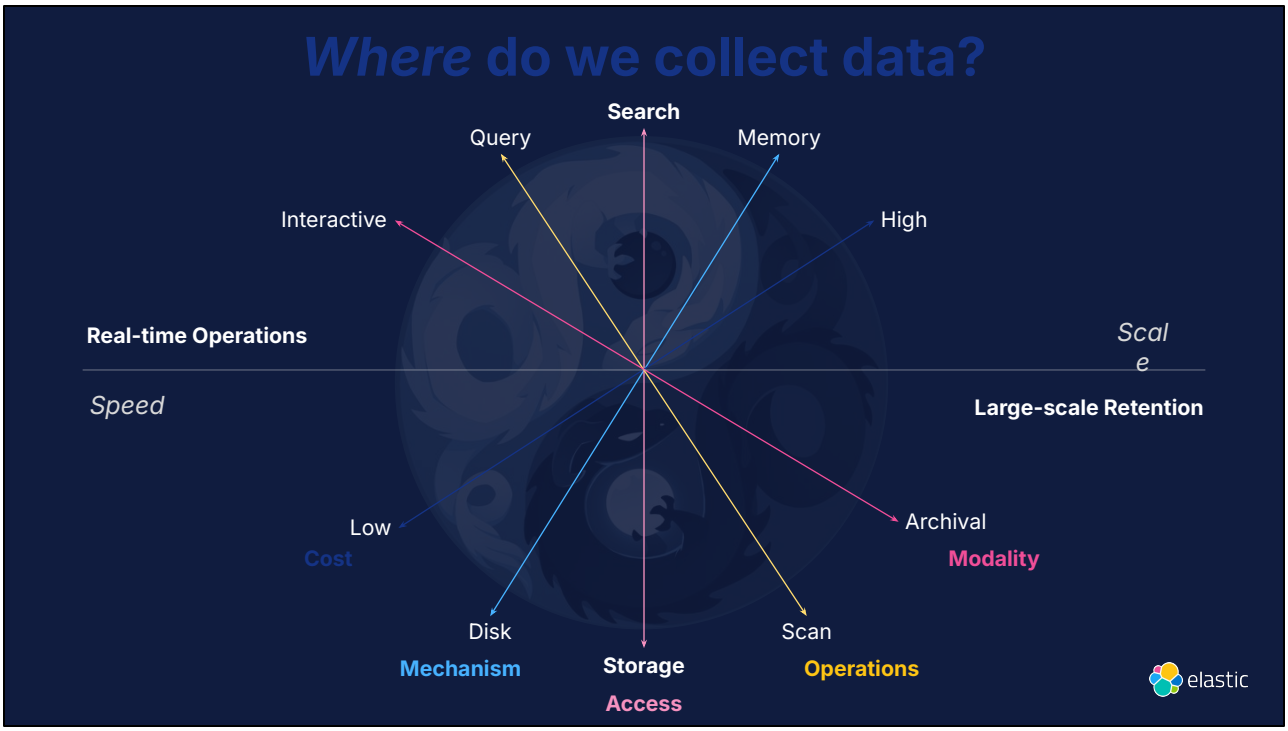
The Challenge:
Data is generated everywhere—across systems, teams, and geographies.
How do you get **total visibility—at speed, at scale, and without breaking the bank?**

—----------

I'd like to start this presentation with kind of an esoteric question: Why do we collect data? The short answer is: so that we can USE it. Data is the lifeblood of all our operations – in order to make good decisions we need to consider all available information, both to see what what's relevant as well as what's not. Some data we need to analyze immediately, and some we need to hold onto just in case it's needed later on. For that "just in case" data we normally want to drop it as quickly as possible into the cheapest storage available.

- But real-time operations requires data to be readily available to retrieve, connect, and compare, and just as importantly we need the ability to interrogate and pivot interactively when our queries surface new clues.

- The problem is that the data we need is generated everywhere… How do we ensure we've considered all data points when making those decisions? How do we achieve total visibility, at speed, at scale, and cost-effectively? Those are the kinds of questions that caused a what I would call an inflection point, it was a decision that has largely shaped our technology development directions since…



That's where we hit an **inflection point** as an industry.
The debate was: **search for speed vs. storage for cost.**

- **Search** is like memory—fast, interactive, lets you pivot quickly.
- **Storage** is like disk—cheap, but slow, requiring batch scans and rehydration.

—--
The decision was about "Where" we should collect data… The choice was between <CLICK> the opposing sides of Search as the access paradigm or scalable cost-effective Storage.

- The differences between the two sides are many, but an easy analogy to remember is <CLICK> the difference between computer memory and disk – Search indices live in memory for fast retrieval, and Storage lives on disk for long term retention.

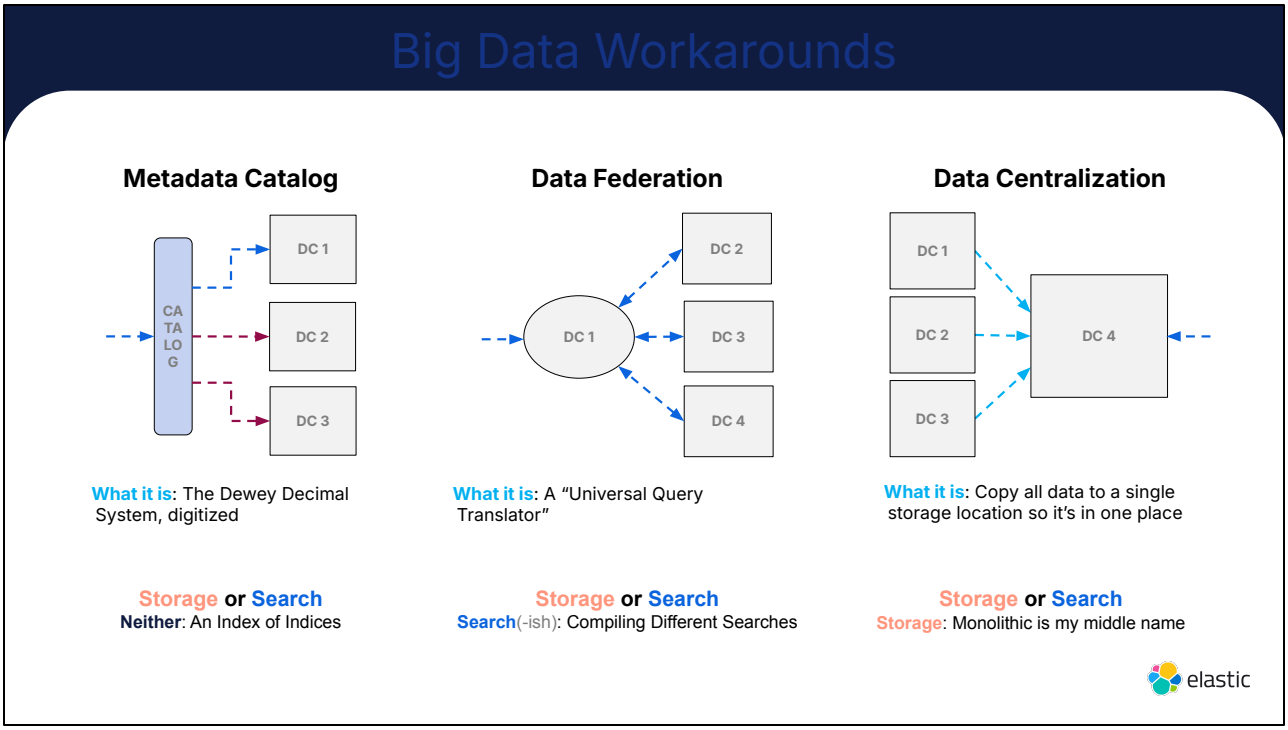
- Because of that <CLICK> we have different operations available on the data – with search we can query it immediately, where on storage we can only scan it.

- And that <CLICK> affects the way we're able to use the data - search allows for interactivity with the data, while storage requires that you rehydrate it first.

- Looking at the scoreboard so far, I'd have to say Advantage Search! So why didn't Search become the de facto storage mechanism for big data collection?

- Until fairly recently, (and I'll explain that in a few minutes) <CLICK> the biggest factor of all was cost: the last time I looked Memory was running over $4 per GB, where Disk was around .12 cents per GB – in the most practical of terms, it doesn't matter how fast or flexible your Search paradigm is, if you can't afford it then it doesn't work.

  I just want to highlight that it's not really an either/or situation – we can and should use the technology best suited to the task at hand, but to my mind that should be determined by how we plan to *USE* the data being collected more than anything else. The cost factor was not just something the DoD had to deal with – this was the path the tech Industry chose.



Cost pushed the industry toward **storage-first architectures**—but that came with trade-offs:

1. **Catalogs** only give you pointers, not insights.
2. **Federated Search** bridges systems but is slow and inconsistent.
3. **Centralization** (Hadoop, S3) keeps data in one place but doesn't solve **interactive access**.

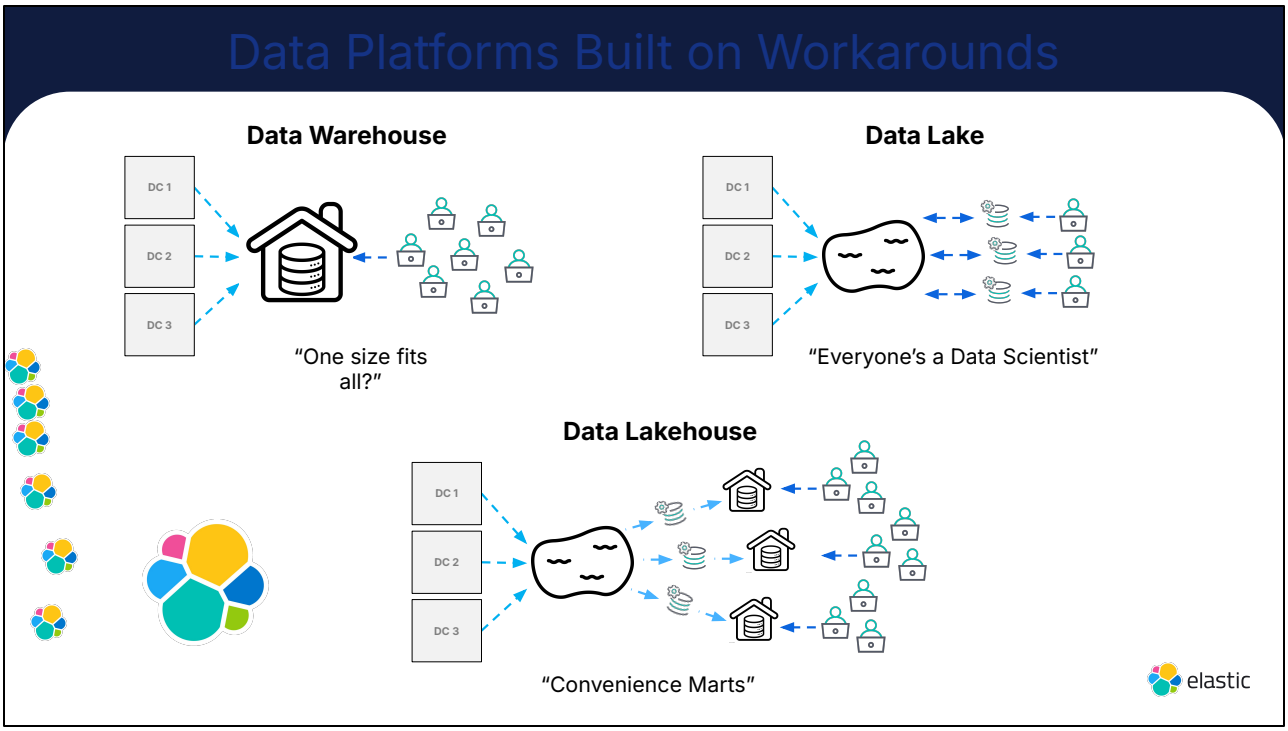
———

As the Industry followed the storage paradigm many technological approaches were tried, but they're all workarounds to that core inability to search distributed data at once. Each approach introduces its own set of problems…

- Catalogs can only operate on a limited amount of discovery metadata to tell you where you might be able to find relevant information.
- Federated Search is a distinctly different thing than Distributed Search – Distributed Search lets you query remote locations using a common platform, whereas Federated Search tries to create bridges between systems

- that use data differently and speak different languages.
- By far, the most prevalent of the three patterns is Data Centralization and that's become the basis of platforms like Hadoop and more recently object stores like S3: they provide extremely scalable low-cost storage where you can scan everything in one place

- Unfortunately, simply having all the data in one place isn't enough – we need to be able to interact with the data, not just store it…



This led to **warehouses** and **lakes**—then hybrids like **lakehouses**—all trying to solve this "store everything, query later" problem.

But they're still built around **scanning storage**, often **limited to SQL**, which isn't fast or flexible enough.

———

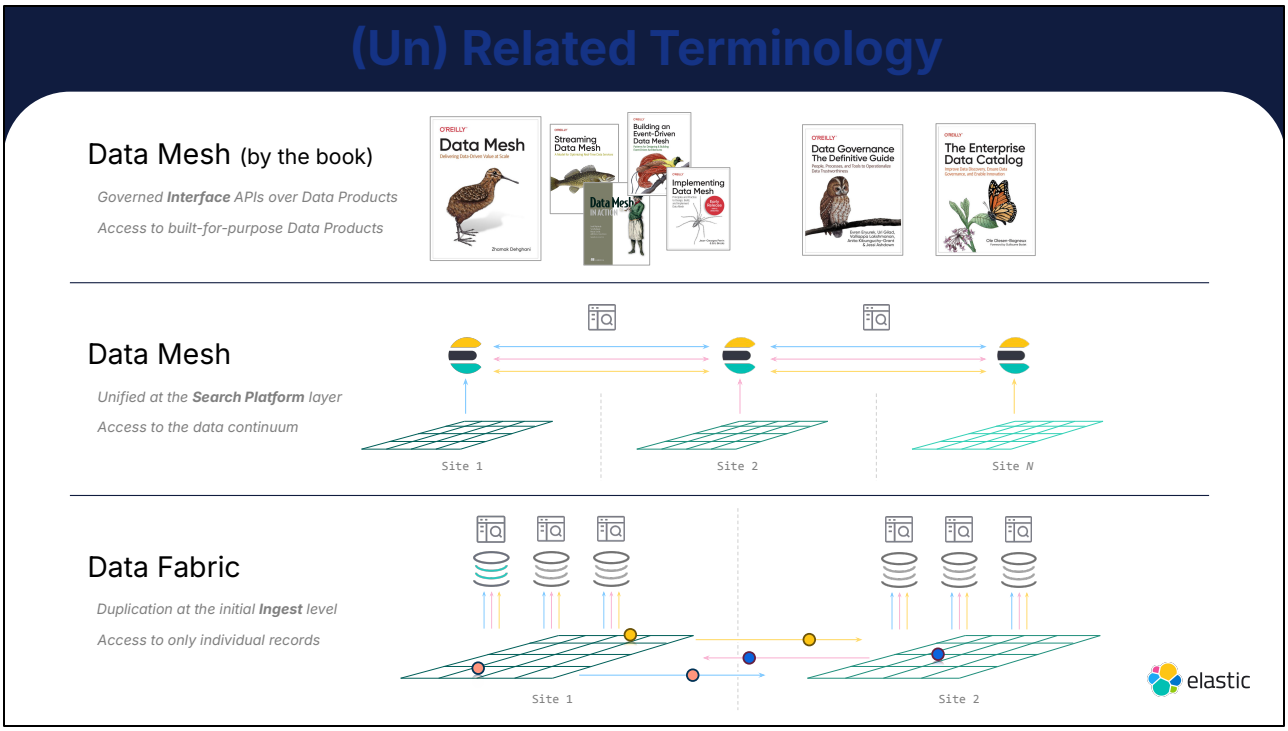
One design that's been around for a long time is the Data Warehouse – a Warehouse tries to create a single data platform with a schema that supports as many use cases as possible, but the more use cases it has to fit, the less flexible it gets

- Many found the number of use cases a Data Warehouse can support is too limiting, so they chose <CLICK> to step back to storing data in a more raw, "it's all here if you need it" kind of approach – also known as the Data Lake. For early versions of the Data Lake the data was collected in a raw state which means that in order to user it, the data needs to be read, reformatted, and ingested into something that can make sense of it. For the most part only Data Scientists know how

- to do all of those those tasks effectively… Newer iterations of the data lake try to side-step some of that by immediately ingesting incoming data into a format like Parquet, which forces all data into a columnar datastore which does not fit every type of data.

- Lately there's also been a hybrid version of the Warehouse and Data Lake – the Lakehouse approach makes smaller purpose-built warehouses available for specialized use cases.

- All of these designs are based on the desire to make a centralized data repository queryable in some fashion, almost exclusively via SQL. As powerful as SQL can be, it can be obtuse and difficult to construct queries, and not especially fast or flexible to use.



Data Fabric vs. Data Mesh:

This is where **Data Mesh** enters—offering a better approach:

- **Data Fabric** copies data between sites, but creates more silos.
- **Data Mesh** lets data stay where it's created but **indexes it locally**, making it searchable across all locations through a common platform.

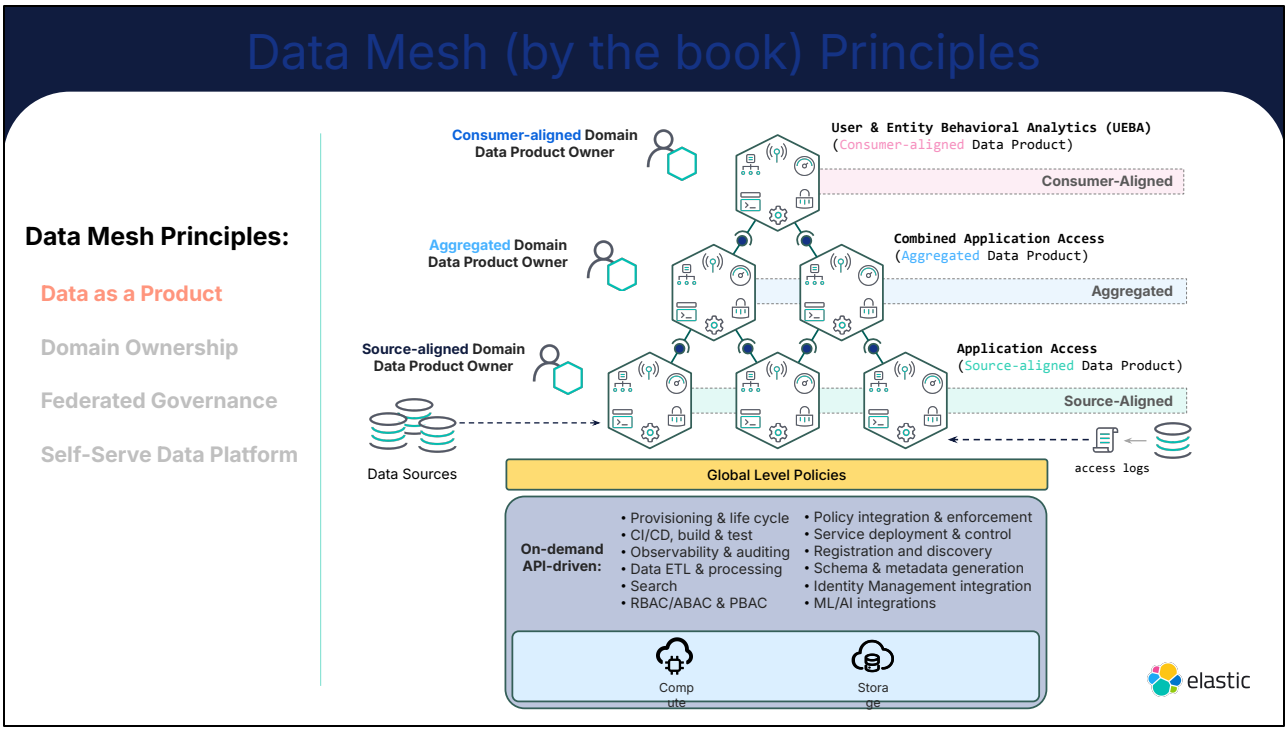
—————

- We're at the point where I should probably define the differences between some seemingly closely-related terms.

- <CLICK> The first one is "Data Fabric" – a data fabric allows data that was collected at one site to be <CLICK> copied to another site. The data shared between sites is still in the form of individual records, it's not able to be correlated with other records until <CLICK> it gets consumed by something that can make sense of it; this is a modern method for creating data silos.

- <CLICK> The next term is Data Mesh. <CLICK> The difference between Data Mesh and a Data Fabric is that data is still collected at each remote site, but instead of getting copied to remote sites it gets <CLICK> indexed locally into a distributed search platform that can search both locally and <CLICK> across remote sites. This is the very pragmatic form of the term data mesh – meaning that we index the data once and it is made available to any consumer or use case through a shared search platform.

-
- <CLICK> There's a new definition for Data Mesh that's gaining traction, based on a book that's now <CLICK> spawned an entire ecosystem around the ideas. This version of Data Mesh is concerned mostly with theoretical constructs about data stewardship and tries to make those ideas practical through recommendations for standardized objects and APIs. Smartly this new conceptualization of Data Mesh includes Governance and Catalogs as part of the design, but as it's written this form of Data Mesh is still based on many of those same workaround technologies we discussed earlier. (In case you're wondering, O'Reilly is a publisher of technical books where all the covers depict weird animals, I don't know why…)



Data Mesh as a Product Approach:
The modern take on **Data Mesh** focuses on **treating data as a product**—owned and managed by the teams closest to it.

Each **Data Product** includes:

- Raw data inputs
- Transformations
- APIs for access
- Security, performance monitoring, and governance

There are **three types of Data Products**:

1. **Source-Aligned:** Cleaned-up raw data (e.g., a log stream).
2. **Aggregated:** Combined data from multiple sources.
3. **Consumer-Aligned:** Final reports or insights for end users.

Federated Governance ties it all together—balancing global policies with domain-level control.

The Self-Serve Platform:
All of this runs on a **self-serve data platform** that:

- Supports ETL, search, and delivery
- Integrates with identity and security systems
- Powers advanced functions like **AI and machine learning**

There are four foundational principles of a Data Mesh as laid out by the book…

- The first principle is the creation and delivery of Data as a Product. <CLICK> This kind of flips the way we currently treat data on its head – instead of collecting raw data into a storage system and letting everyone find and figure it out for themselves, this principle says that each domain should treat <CLICK> its data as a high-quality, well-documented, and easy to use Product for others to consume. A key word there is “Domain"...

- <CLICK> Each domain within the organization knows best what data it needs to accomplish its mission, so Data Mesh assigns responsibility for the creation and ownership of Data Products to each Domain. This means they are maintained by <CLICK> Data Product Owners who have a vested interest in assuring data quality for the use cases their data is being delivered to. Of course, with great power comes great responsibility and that means the Data Product object has to contain everything it needs to deliver a high-quality product: <CLICK> it needs to know to consume raw data inputs, it includes the code to transform and normalize the data, how to compose the data product and all the related metadata into consumable assets; it knows how to serve those assets as outputs via APIs; it knows how to monitor performance and report usage on the assets, and also includes domain level policies controlling security and governance of the Data Product. It’s a LOT.

There are three main types of Data Products:

- Source-aligned Data Products are closest to the raw data they’re based on; so think of an individual type of log stream, or a database feed, etc. A Source-aligned Data Product is basically a cleaned-up and governed deliverable of that original data source.

- Data Products can build on top of each other, and the output of one Data product can be an input for another Data Product… That means a <CLICK> Domain Product Owner can create an Aggregated type of Data Product, and it does just what it sounds like: it's an aggregation of multiple inputs perhaps with some additional filtering or normalization applied to make the data suit their Domain's needs.
- <CLICK> A Consumer-aligned Data Product usually represents a very highly-tuned deliverable that is the result of consuming and applying human-level understanding on the data; the output could be something like a generated report summarizing an opinion on what it all means to the reader, or maybe something much more complex.

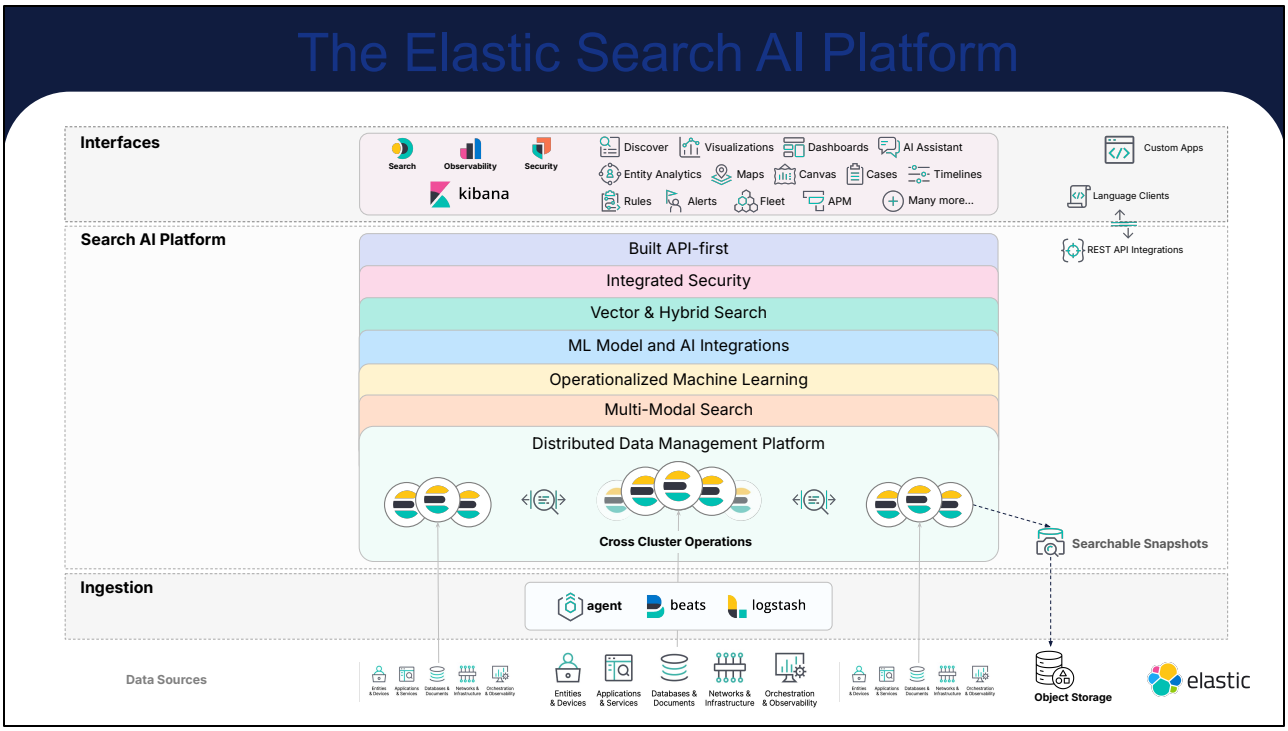
  To illustrate with an example, let's say we have an application <CLICK> that we want to be able to report access on , we can create a Source-aligned Data Product on that log stream. We could then use that Source-aligned Product as one of the inputs for an <CLICK> Aggregated Data Product that combines access logs from across the organization. That Combined Access Data Product would definitely be the type of input we'd want to include <CLICK> in a Data Product built to compile User and Entity Behavioral Analysis activity.

- The third principle is Federated Governance: This one's pretty simple: Global governance policies are defined through a committee comprised of data platform owners and Data Product owners, and those global policies get inherited and integrated with additional domain-level policies, that are then enforced within each Data Product

- Data Products all live and work together in the "Self-Serve Data Platform"… The Self-Serve Data Platform knows how to provision, build, and deploy new Data Products (including all the functions of the Data Product); it supplies shared resources for performing functions like ETL, search, and

- delivery services; the Platform knows how to integrate with external data sources like identity management systems, and policy engines; oh and it also has integrated Machine Learning and AI capabilities as well.

That's quite a LOT to expect from a single platform (and I'd say that the Data product concept is overloaded too), and it's made more difficult by the assumption that all of this overlays a data lake, where again the primary function is storage and the only way in is through SQL!



Elastic has been building this for over a decade.

The **Elastic Search AI Platform** is uniquely positioned to power this vision—combining search, analytics, and operational visibility into **one platform** that scales with you.

———

-
- Here's the good news: Elastic has been building that platform for over a decade! The Elastic stack is one product that spans all three layers, including Ingestion and Interfaces, but the heart of the stack is the Search AI platform

- Elastic provides a variety of data collection, ETL, and ingestion tools to bring any type of data into the platform...

- The main feature of the platform is Distribution – a single instance of Elasticsearch scales horizontally to form a cluster that leverages parallel compute resources to provide data resilience, scalability, and internet speed search.
  An Elastic cluster can be deployed anywhere and everywhere, including on-premises, in the cloud, in virtualized, or orchestrated containers and they all operate exactly the same way.

- There are two main capabilities that make Elastic especially suited for Data

- Mesh: Cross Cluster Operations give you the ability to connect remote Elastic clusters and query them as if they were in the same cluster, and Searchable Snapshots make long-term data retention cost-effective. Those two capabilities together are a BIG DEAL: CCS gives you the ability to search, in near real-time, across the data mesh without having to copy it all to a monolithic repository (and you get to make use of both the search paradigm and parallel compute), and Searchable Snapshots let you keep an active compressed search index for multiple years of retention and potentially hundreds of Petabytes of data that sits in cheap object storage.

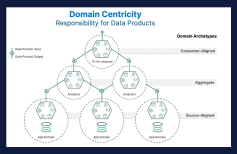
- Included in that unified platform are a host of fully integrated features that most other platforms have to bolt on, things like: best in class search and aggregations; integrated unsupervised and supervised machine learning, including the ability to import and apply ML models onto your ingested data. BTW Elasticsearch is also the most popular vector database in the world, and we can combine vector search with all of our other search modalities to provide unique hybrid search capabilities. One of the hardest things to manage in a federated system is aligning the disparate security controls, Elastic provides integrated RBAC and ABAC security that can be applied to assets at multiple levels.

- The entirety of the stack is built API first, and we provide client libraries so that you can use the Elastic platform to power really any operational data system. Those same APIs are also the bridge to Kibana which includes several market-leading solutions for Search, Observability, and Security as well as fully customizable visualizations, dashboards, data management, and analysis tools.

Elastic operationalizes the Data Mesh

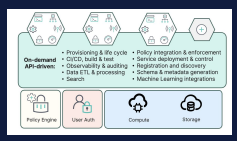
Elastic provides an enterprise-grade, search-based, distributed data platform

- Search can be the foundational mechanism for 'Data Products'
  - ☑ Inputs = Indexing
  - ☑ Outputs = Queries
  - ☑ Observe = Monitoring
  - ☑ Code = ETL Processing
  - ☑ Compose = Data Views
  - ☑ Governance = Filters & Security



- 'Data Product' archetypes align directly to distributed search capabilities
  - ☑ Source-aligned
  - ☑ Aggregate
  - ☑ Consumer-aligned

- A unified data platform gives consistency to applied Governance

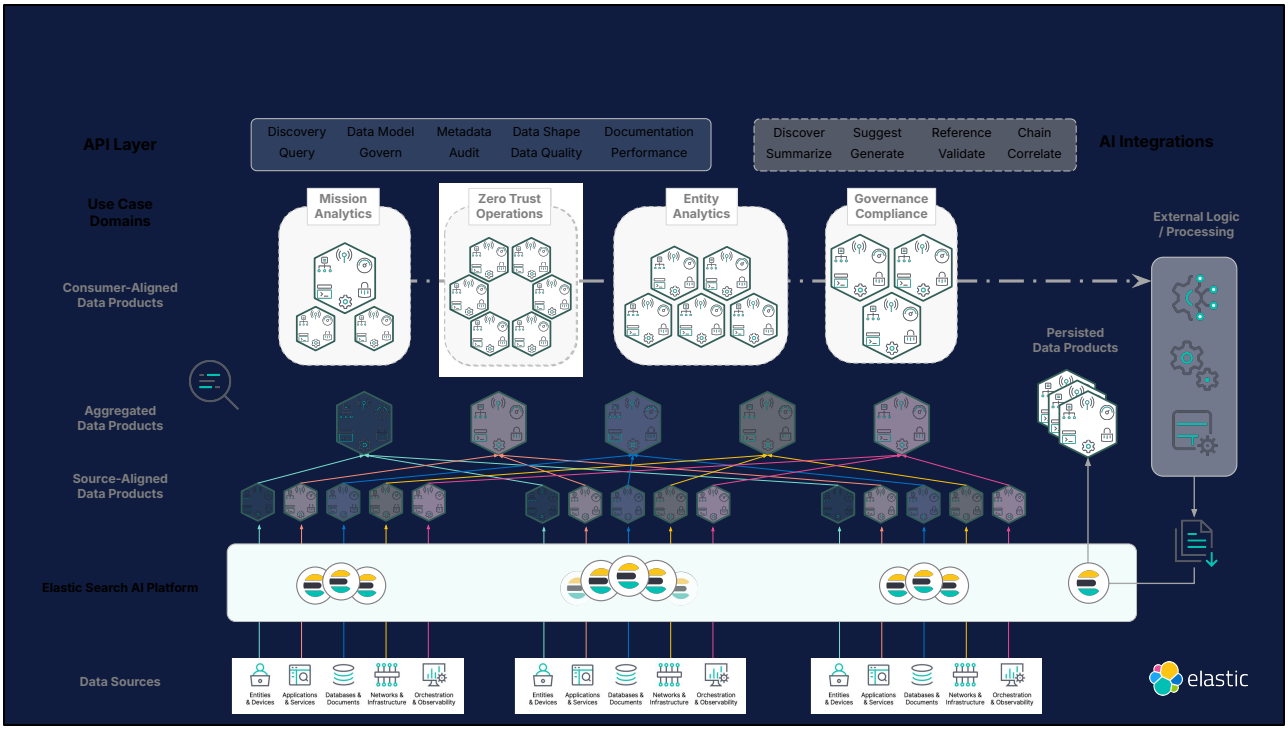


- Elastic puts the data platform in 'Self-Serve Data Platform'
  - ☑ Scalable Data Platform
  - ☑ Security & Governance
  - ☑ REST API Integrations
  - ☑ ETL & Normalization
  - ☑ Search and Discovery
  - ☑ Distributed Operations
  - ☑ Provisioning & Life Cycle
  - ☑ Integrated Machine Learning
  - ☑ Platform Agnostic

elastic

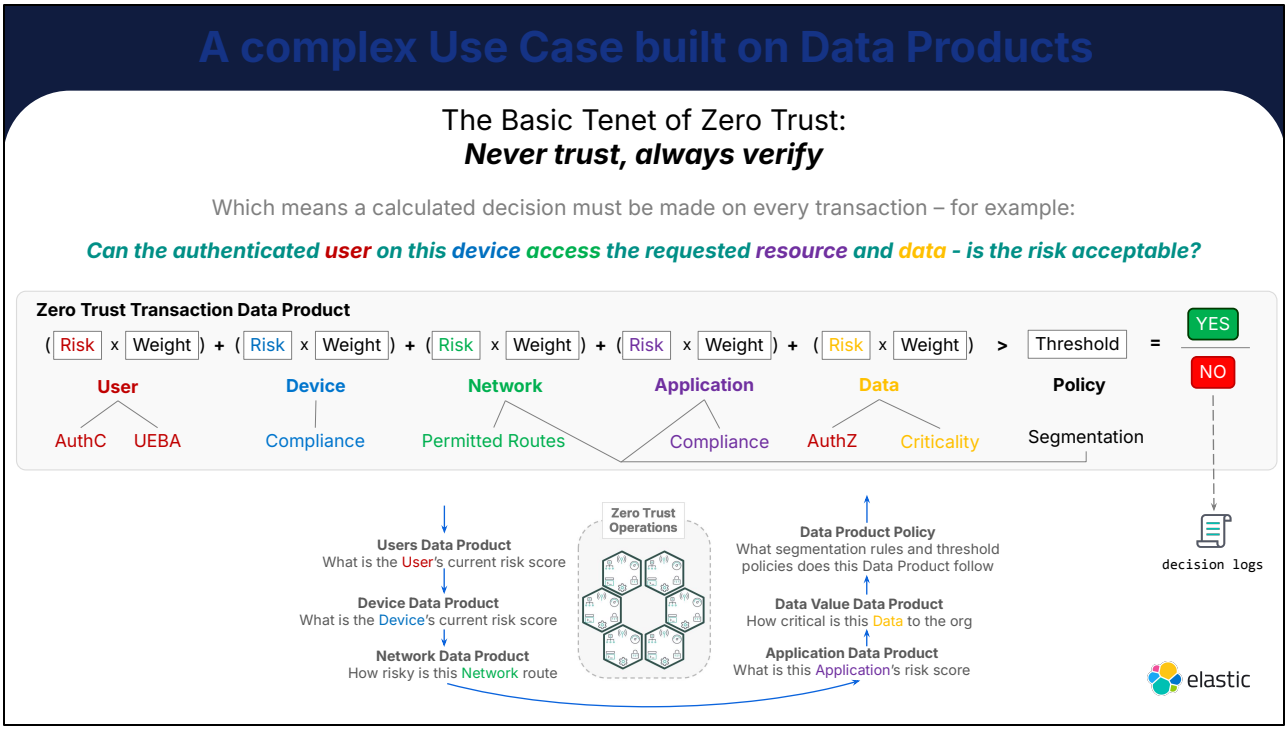
The Elastic platform covers that the majority of the core requirements and I'd say a large percentage of the stretch goals that Data Mesh (by the book) is attempting to define – with that platform you get:

- The ability to ingest and perform ETL on any type of data, store and query that data in a flexible manner, and to apply security and governance on those indices directly within the platform.
- Source-aligned Data Products are essentially indexed data streams, and the Aggregation of those is accomplished through built-in functionality like Data Views and Cross Cluster Search
- A single data plane greatly simplifies the creation and application of security and governance rules.
- And Elastic provides the core features of the Data Platform itself, it's already built to be flexible and scalable, without the need to rely on bolt-on capabilities.



So now if we put it all together…

- Instead of funneling and copying data from around the world into a data lake, we can keep data near where it was generated <CLICK> and index it into a distributed search platform, where the act of indexing itself creates “Source-aligned” data products.
- With Cross Cluster Search, these <CLICK> naturally become the next layer of “Aggregated” data products – and so far we have two levels of Data Mesh Products powered solely on the core search capabilities of the data platform.
- From these search primitives <CLICK> you have a foundational speed layer to build your “Consumer-aligned” data products on. For any Data Products that require human-derived logic or external processing, <CLICK> the platform’s API-first nature allows it to be the data engine feeding those external processes, and the results of those operations can be stored back into Elastic to become interactive data products that power <CLICK> the organization’s mission-critical Use Cases.
- <CLICK> Those use cases in turn can have Data Product APIs layered over them so that they can also be monitored and measured and governed.
- Finally – with its integrated support for vector and hybrid search, we can continue to innovate and use Generative AI and RAG techniques to leverage our Data Products in exciting new ways.



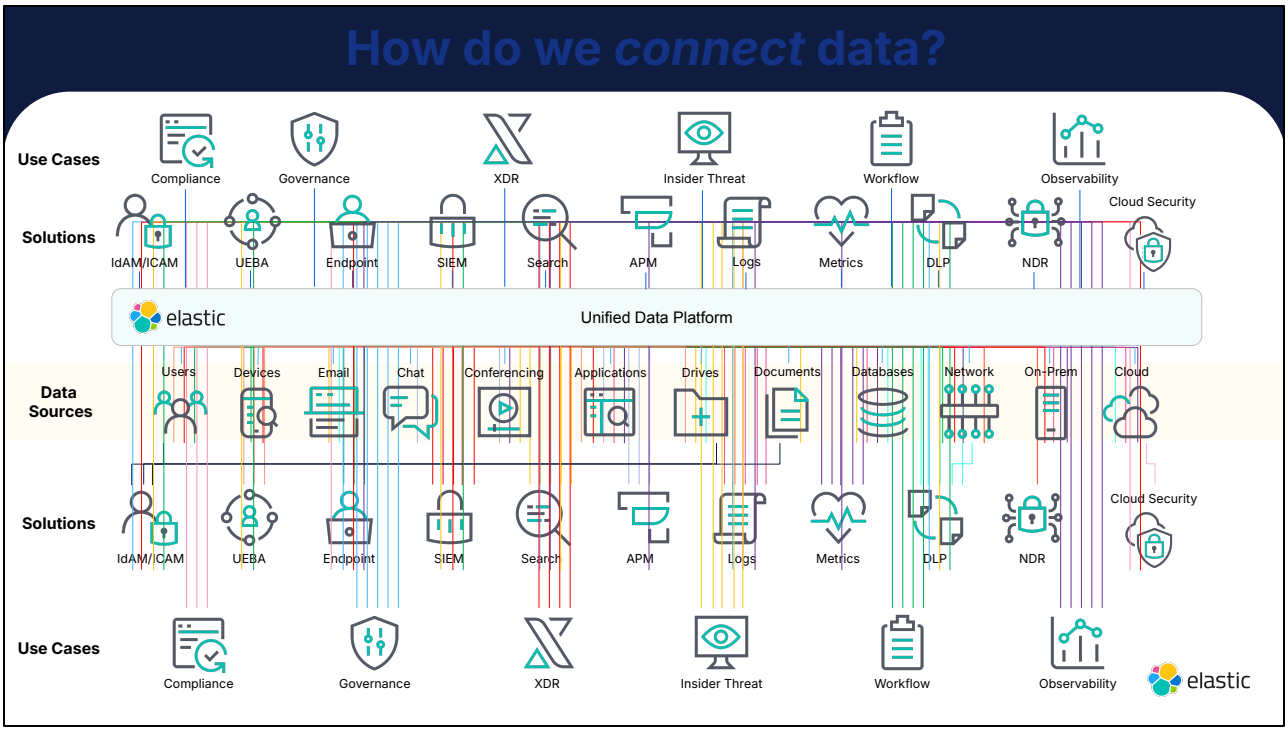
- Let's build out an example... We can create a Data Product on top of a Use Case like Zero Trust that needs to perform various complex steps all together as a logical unit –

- For example, let's say we want our Zero Trust Data Product to be able to answer questions like this: That single question is actually several queries, and within each of those are multiple sub-queries… Let's break it apart and create high-level Data products for each of those items, starting with the formula that we defined in the policy component of our Zero Trust Data Product to decide whether the transaction meets the necessary criteria to be allowed.

- Let's start with a Users Data Product that we can query to return the current risk score for the user involved. That risk score will be based on several inputs, including things like the user's role and behavioral analysis.

- From there we request the risk scores for Device compliance and Network segmentation, and check the Application's vulnerabilities, and weight all of that against the criticality of the Data being requested and we plug those values into the formula defined in the Zero Trust Data Product's policy to determine whether or not to allow that transaction to proceed. Magic! <CLICK> Another aspect of unifying these items under a Data product is the ability to monitor and then audit the decisions that were made.

- Zero Trust requires the ability to ask multi-layered questions in a near real-time manner – Zero Trust can't be built on individual point to point integrations between products or layers: the only commonality and the only way to unite all those systems is through the data they produce. In my opinion, you can't achieve Zero Trust without a unified platform like Elastic that can make all of it immediately queryable.



- And that's another reason why a Distributed Search platform is a superior approach – it's in how we connect our data. All organizations have some combination of different types of business systems built to collect or leverage different types of data in different ways, and for different purposes…

- When we go to perform system-spanning operations like Security, or Observability, most solutions are built to integrate to each system individually. Even if you could restrict everything to be all in the same ecosystem, the different integrations for each use case are usually directly to each individual touchpoint, they're essentially bolt-ons to each system and are made to work with one type of data.

- And there are multiple layers of this, with higher-order use cases connecting to multiple solutions, but again as point solutions

- Just like it is for Zero Trust - Data is the common denominator in all of these different systems - if you can instead ingest it ONCE into a single search platform, it becomes available

- To power all of your existing use cases in a much simpler and interconnected way, and for any other future use cases you can dream up. This is Data-as-a-Service!

- (Elastic stores data more cost-effectively (by orders of magnitude), and keep sit in a searchable, interoperable state throughout its life cycle)

The Elastic Search AI Platform is an *operationalized* Distributed Data Mesh

- An open data platform that unifies data access and integrations
- A scalable, cost-effective platform to keep data interactive
- Spans all deployment platforms and data types
- Provides flexibility and speed to data operations
- Elastic delivers unified Data-as-a-Service

elastic

- Elastic provides the distributed Search paradigm that we need for real-time operations
- With Searchable Snapshots we can keep data in an interactive and searchable state for as long as we need to, and do so cost-effectively
- We can connect data across platforms, across systems, and across clouds and on-premises
- We can step out of the limited world of point-to-point integrations that only allow you to interact to a few data sources at once, and give you the flexibility to build data products for any and every use case
- This is how Elastic delivers true Data-as-a-Service to all of your operations through a Distributed Data Mesh.


Thank you
elastic