Yellowbrick

WHITE PAPER

The case for a new Cloud Native Data Warehouse

Background

The first attempts at cloud data warehouses were born around 2011-2012. Amazon Redshift and Google BigQuery took very different approaches to trying to bring the world of data warehousing to the cloud.

Amazon licensed legacy technology from ParAccel, an ill-fated start-up founded in 2005, and shifted it into cloud-based instances that users could provision online. Although Amazon has continually improved the codebase, the clunky architecture still manifests itself today, forcing users to deal with the complexities of cluster management and missing key features such as disaster recovery. Google built a simple query engine to allow users to analyze web-scale data. It was built on Google's proprietary infrastructure, started with on-demand billing and storage/compute separation, and has since evolved into a credible but complex data warehouse offering that still struggles with complex schemas and workloads.

Snowflake was founded in 2012 and made use of the best possible features of cloud architecture from the time: Instant on/off, data persisted on object stores to separate storage and compute, multiple clusters managed through SQL, ease of provisioning and use, large volumes of compute, and on-demand billing. It's grown into a massive SaaS platform in its own right, with a large number of locked-in services built around the data warehouse, but the core data warehouse is incredibly simplistic and struggles with highly concurrent, ad-hoc workloads.

Wind the clock forward to 2015, the most important shift in Cloud Native architecture took place: Kubernetes – the container orchestration platform – was released, and the Cloud Native Computing Foundation was formed. Cloud Native architecture was developed to allow enterprises to build and deploy software that avoids cloud vendor lock-in, truly enabling "cloud software that runs anywhere." Modern Cloud Native applications are built using container architecture and composed of orchestrated microservices. They run on generic infrastructure, are monitored and managed in standard ways, and authenticate using open protocols.

Cloud Native re-architecture for scale-out web applications is commonplace however databases have mostly just been "lifted-and-shifted" into the Cloud Native world: Plonking a database into a container allows it to run in modern infrastructure, but it doesn't give the modern Google and Snowflake user experience: The software is largely ignorant of the fact that it's running in a container environment, and operations such as managing elastic clusters (which Snowflake calls "virtual warehouses") have to be managed from outside using "Helm charts" for installation/upgrade and "operators" for management – complex concepts that are only familiar to Kubernetes admins. Features such as allowing multiple, elastic on-demand compute clusters to share the same underlying stored data are unavailable. Users seeking to get business value from an elastic, cloud-based data warehouse don't want to know about "Helm charts," pods, nodes or configuration files. They just want to provision data warehouses, manage elastic clusters and gain insights from their data.

Yellowbrick is the first truly enterprise-class data warehouse to re-architect for modern cloud native architecture: We deliver separate storage and compute with elasticity managed through SQL, combined with the best features of enterprise-class data warehouses like support for high concurrency and workload management… all tightly integrated with Kubernetes, running anywhere.

Requirements of a modern Enterprise Cloud Native Data Warehouse

When considering the case for a new Cloud Native Data Warehouse, we need to consider the needs of enterprise customers who rely on the data warehouse for essential operations. The requirements are as follows:

Modern, elastic architecture: Elasticity, separate storage & compute and ease of use form the foundation of a modern data warehouse.

Act like a database with the basics: Enterprises don't want to hire software developers to keep the warehouse running, and deal with the regular crashes that data lake vendors can't fix. Ad-hoc workloads, security, transactions, high concurrency, workload management, and disaster recovery aren't optional.

Run across all public clouds: Enterprises don't want to be locked into one cloud vendor forever, and the preferred vendor can change over time. The data warehouse must run on AWS, Azure and GCP, and be extensible to support public clouds in other regions, to avoid lock-in.

Run in VPCs and SaaS: Larger enterprises want to keep their data in their VPCs and pay their own infrastructure costs, rather than sending their data to someone else's SaaS, for regulatory, compliance or affordability reasons.

Run hybrid: Larger financial services organizations have on-premises deployments for regulatory, compliance or historical reasons. Other leading edge cloud companies have private clouds in their own data centers to save money. The same data warehouse must run in all these environments.

Run on generic infrastructure: Many deployments don't require optimized hardware instances; from laptops to OpenShift environments, the data warehouse must adapt to the capabilities of the available hardware instances.

Predictable pricing: Purely consumption-based models provide no incentive for the data warehouse vendor to improve performance and concurrency — why improve throughput if it leads to lower revenue? Enterprises that budget ahead need to know their costs; a modern warehouse must support consumption-based and predictable, budgeted pricing models.

Support a service provider model: Many large enterprise data platform teams run services on behalf of lines of businesses. Others really are true service providers. This requires an isolated multi-tenant management architecture, where the data platform team can provision and manage instances on behalf of lines of businesses and bill them back accordingly.

No current cloud or on-premises data warehouses can satisfy these requirements. The remainder of this paper discusses Yellowbrick's architectural approach and how we satisfy these requirements.

Unified management for the service provider model

Data platforms are a resource provisioned by a centralized IT organization, where usage is either billed back to lines of business or to outside customers in a multi-tenant model. At Yellowbrick, we embrace this model: Our customers first interaction with our product is to provision instances using our Cloud Data Warehouse Manager (CDWM).

CDWM is a management tool, with a simple user interface and GraphQL API, that provisions data warehouse instances on behalf of either lines of business or outside customers. CDWM can place different instances in different clouds or on-premises. Usage of the instances is metered and can be billed back accordingly. Each provisioned data warehouse instance is itself elastic, with separate storage and compute, allowing spin-up and spin-

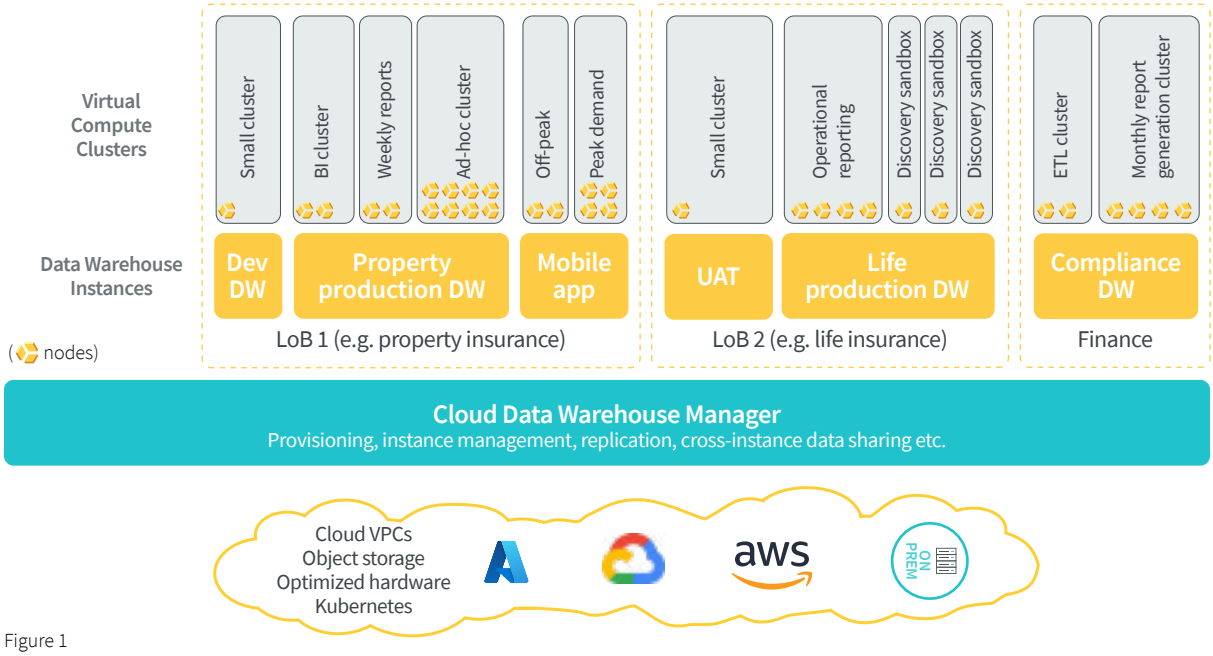

Figure 1

down of compute clusters as described above. Resource consumption limits, where needed, are placed on the instances by the CDWM administrator. This architecture is shown in Figure 1.

This model extends beautifully to service providers: For the first time, service provider businesses that wish to offer a multi-tenant, elastic data warehousing service can now do so easily, running either on public cloud infrastructure or inside their own data centers.

We accomplish this without a shared metadata architecture: CDWM itself is not essential for operation of individual data warehouse instances and there is no central point of failure, making global outages experienced by some SaaS data warehouse providers impossible with Yellowbrick.

Separate storage and compute

Separate storage and compute are the hallmarks of elastic architecture. Although many vendors are now touting this functionality, not all of them have the same level of capability. We define a level-based taxonomy to separate the marketing hype from the reality:

Level 0: No separation; storage and compute are co-located, like traditional MPP warehouses. Expansion is often possible (although many legacy systems require downtime) but elasticity (on demand grow/shrink) is not possible.

Level 1: Data is stored separate from compute nodes on high cost, high performance storage (EBS, SAN, NAS or custom storage); storage can be resized online, but resizing clusters is still an expensive operation.

Level 2: Data is stored separate from compute nodes on low cost, low performance storage (S3-compatible object stores) and cached locally. Resizing is clunky and provided via management tools, and only one elastic compute cluster is supported.

Level 3: Improves on level 2 by allowing multiple elastic clusters to access the same storage.

Level 4: Improves on level 3 by allowing new clusters to be easily provisioned, suspended and resumed through SQL, or automatically according to workload demands, with no special constraints.

Level 5: Improves on level 4 by supporting high performance API-based storage access (load and simple query) without requiring compute clusters and – in the presence of sufficiently fast networks –

Here are some examples of different levels supported by different vendors:

Level 0	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
IBM Netezza	IBM NPS	Azure Synapse	Vertica EON	Snowflake	Google BigQuery	Yellowbrick 2022
Teradata	Oracle		Redshift RA3	Yellowbrick		

with no caching. No caching means instant response times for starting and stopping clusters. API access means data loads and access to stored data via Spark data frames can be accomplished without the cost of running a running compute cluster.

Level 6: Improves on level 5 by allowing access to on-premises and cloud storage systems, with a variety of protocols including object storage, NFS and others and allowing placement across different clouds and data stores.

At Yellowbrick, we have recently implemented Level 4 separation and are in the progress of implementing Level 6. Our implementation is unique in several different ways, catering to both the hybrid cloud nature of Yellowbrick as well as its heritage in enterprise applications:

Hybrid flexibility: Yellowbrick supports storing data locally, for the fastest possible performance without caching, as well as cached on separate storage. This means that on-premises data warehouses can enjoy the benefits of separate storage and compute.

Protocol flexibility: In addition to all the major S3-compatible object stores, at Yellowbrick we also support NFS storage devices as the separate storage tier for flexibility and choice of storage vendor. In some workloads, NFS offers much higher performance than object storage.

Granular placement: Data is stored in our customers' own S3 buckets, meaning that Yellowbrick does not charge storage costs. Placement can be configured at the database, schema or table level, to allow large databases to be migrated between different clouds or storage systems.

These benefits mean that Yellowbrick offers the most flexible approach to separate storage and compute in industry: Our customers pay their own storage costs, place data in whatever cloud or on-premises storage system they like, and don't have to struggle with unpredictable response times when caching large data sets.

Full elasticity via SQL with Kubernetes

Our data warehouse is both multi-tenant and fully elastic. Elasticity is present within each Yellowbrick data warehouse instance. Multiple, elastic compute clusters can be created on demand, all provisioned through SQL. Different users can be assigned to run workloads on different compute clusters, and the compute cluster in use can be changed at runtime through SQL, subject to permission. Clusters can be configured to automatically suspend after an idle period and spin back up again on demand.

For example, a separate compute cluster could be created to run ETL processes when needed, one for ad-hoc BI, and multiple data science clusters. Compute clusters can be expanded online during periods of heavy usage, or turned off during quiet periods to save money. Clusters can be created to run daily, weekly, or monthly batch reporting tasks that are only active during those time periods. Both the size of the nodes in the compute cluster, as

well as the number of nodes, are controllable, and limits on resource consumption can be established at the instance level for predictability. Similarly, it's possible to set up a low-cost replica system that receives replication traffic from a primary data warehouse instance, which can then be scaled-up on demand when the replica needs to be used.

We've implemented this elasticity by not just deeply integrating with Kuberentes, but by using SQL itself as the "user interface" for creating, suspending, resuming and managing clusters instead of developer tools like Helm. Kubernetes is the authoritative source of truth for the state of all clusters. System views showing the state of the clusters source their data from Kubernetes using its APIs. When cluster management SQL statements are entered, the Yellowbrick Data Warehouse reaches out to Kubernetes to change the desired state of the instance; Kubernetes then affects the necessary changes. If a node in the cluster becomes unhealthy, Kubernetes will bring a replacement online.

This is a unique, inside-out relationship with Kubernetes: Rather than Kubernetes being the 'user interface' for driving the state of the cluster, the database – which is managed by Kubernetes – itself becomes the user interface. It's a symbiotic relationship that lets us deliver a world first – making the power and cross-platform flexibility of Kubernetes available to a data warehouse, driven entirely through SQL.

Availability and security

Since Yellowbrick is an Enterprise Data Warehouse, we have designed for high availability of data warehouse services from day one. Although our core database instances are internally highly available, we need to make sure that the data warehouse service remains available in the event of clouds going offline, loss of connectivity, or new bugs being introduced. There are three key things to consider when evaluating the availability of your cloud data warehouse:

Metadata architecture: Yellowbrick's centralized component – the Cloud Data Warehouse Manager, or CDWM – is not required for data warehouse instances to be operational. There is no single point of failure for metadata management or control. Contrast this to the global outages of cross-cloud data warehouse vendors which require centralized services to be available for instances to function.

Durability of data: In Yellowbrick, all data in the cloud is persisted redundantly on replicated cloud object stores or NFS storage. Other storage vendors, such as our partners that offer 5-way global redundancy and optical archiving, provide further assurance to make sure that even events of an apocalyptic scale will not result in data loss. Instance metadata is written to mirrored, EBS-class block storage. For on-premises instances, data can be persisted on external storage, like in the cloud, or on Yellowbrick's own erasure-coded persistent storage.

Replication between clouds: In the event of a global outage of a cloud provider, such as the disruptions experienced in the past by multiple regions of Amazon S3 becoming unavailable, Yellowbrick can run active replication of a data warehouse between clouds from the same or different vendors – or between on-premises data centers and clouds – to make sure cloud provider failure does not lead to an outage. Replication in Yellowbrick is a continuous process with no manual activities required and supports failover and failback.

Both Yellowbrick instances and CDWM authenticate users using standards-based OpenID Connect. This allows users to authenticate out-of-the-box against Identity Providers such as Office365, Okta, and others that implement the standard. Yellowbrick leverages this integration to greatly simplify database-user provisioning and management. In the event of OpenID connect being unavailable, integration into LDAP/Active Directory is available. All data at rest is encrypted. The data warehouse also supports a column-specific encryption technique with externalized key vault, role-based access control and everything else expected of an enterprise data warehouse.

By running data warehouse instances inside of enterprises own VPCs or data centers, data stays local to that VPC or data center and can be secured and firewalled using customers' own best practices.

Maintaining predictable, low cost and optimal performance

At Yellowbrick, we pride ourselves in building the most economically affordable data warehouse, with predictable costs. We do this by offering the best performance per dollar spent in the industry. We have invested extensive effort into implementing the Yellowbrick kernel and our adaptive cut-through architecture on cloud platforms. Some of the optimizations include:

Network optimization: We created a new network protocol, YRD, designed for cloud instances, using Intel DPDK technology. This bypasses the Linux kernel, lowering CPU utilization and driving higher network bandwidth than built-in options.

Direct hardware access: Our data warehouse can directly map attached NVMe SSDs to provide high throughput, low latency access to cached and persistent storage, again bypassing the Linux kernel as outlined in our database architecture white paper.

Custom object store access: We have built new, cross vendor libraries to allow access to cloud object stores with higher throughput and less CPU overhead than vendor-supplied libraries.

Object store file format: We've optimized our storage file formats to allow efficient, highly parallelised transfer of related, minimal amounts of data from cloud object stores, meaning that ad-hoc queries suffer less penalties than in competing cloud data warehouses.

As outlined in our database technical white paper, our 'adaptive cut-through architecture' will apply these optimizations on hardware instances where they are available, even when running in containers. When these optimizations can't be applied, we will fall back to more conventional, lower performing approaches.

With these innovations, for the first time we've created a cloud data warehouse that offers the best performance and economics in industry, making optimal use of all available hardware instances, combined with the ease of use of the latest elastic SaaS data warehousing platforms.

Summary

We've shown how a new data warehouse architecture is needed, enabling:

- A Kubernetes-based architecture that runs across different public clouds
- A data warehouse that runs in enterprises' own VPCs, allowing them to pay their own infrastructure costs
- Runs in hybrid, on-premises infrastructure
- Enables a service provider model for lines of business or external multi-tenant customers
- Users to enjoy a fully elastic, modern cloud data warehouse user experience

Yellowbrick has delivered all the above in its new Cloud Data Warehouse, in the process leapfrogging the traditional, outdated architectures of Snowflake, Amazon, Microsoft and Google.