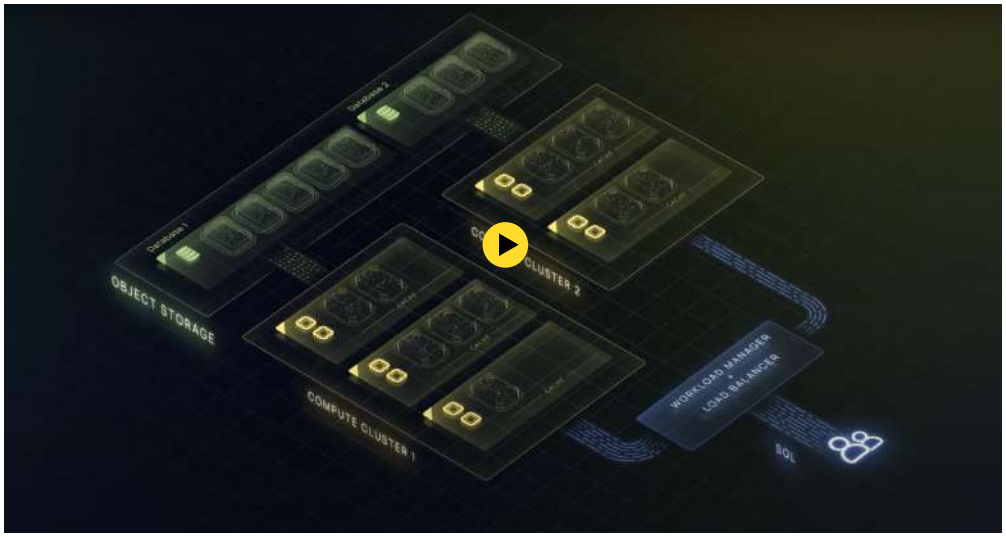
Yellowbrick

How Yellowbrick Manages Concurrency and Scaling with Minimal Infrastructure Needs



Yellowbrick enables cost-effective scaling for B2B applications, handling large data volumes, thousands of users, and hundreds of concurrent queries without the costly, complex workarounds required by traditional scale-out SQL databases.

Don't Take Our Word For It: See How Our Customers Benefit

insightsoftware

Angles Enterprise for SAP simplifies access to supply chain data and delivers actionable insights. By switching to Yellowbrick from their existing data platform on AWS, they unlocked unparalleled data load and query performance on massive datasets, enabling faster insights. The move saved them $40K per new customer onboarded, as their previous platform required costly per-customer instances for data isolation. Yellowbrick's Postgres front-end made data migration effortless, seamlessly integrating with their AWS data stack.

Learn More →

PROOF POINTS

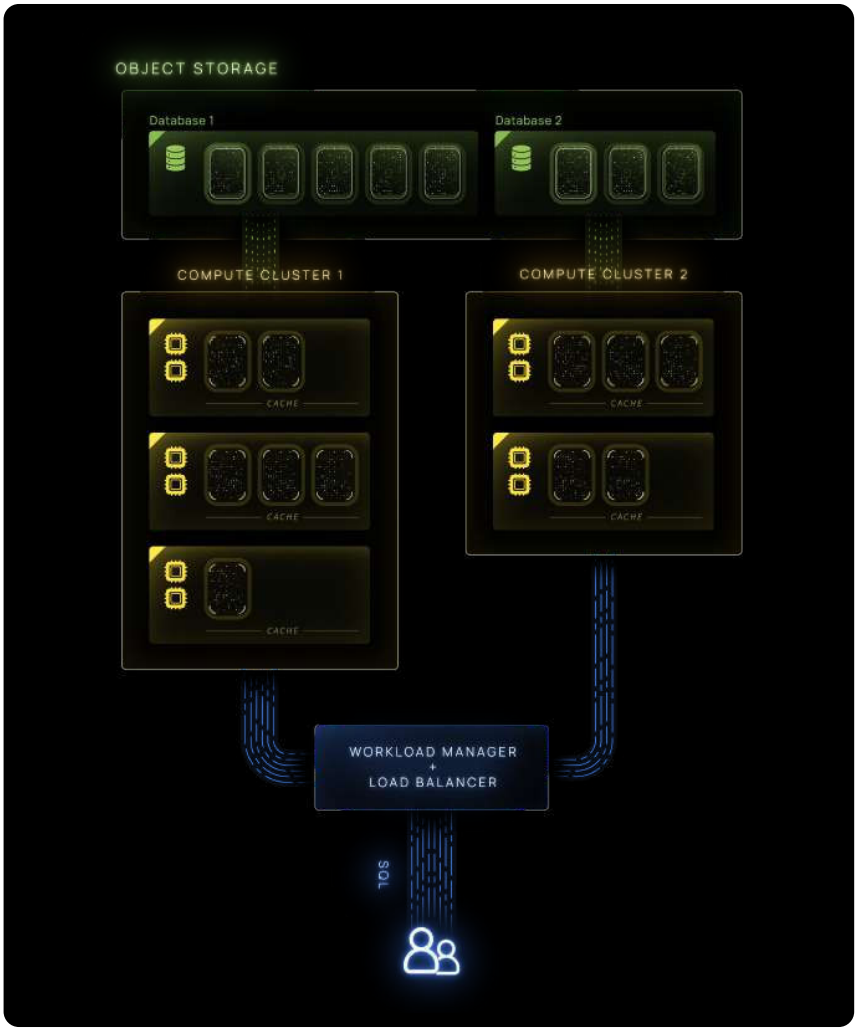
Purpose Built For Scale

Sharing users' data back via embedded analytics is a core feature of most applications. For applications which need to process large volumes of data or service large numbers of customers, this can be a daunting challenge. In any multi-tenant service, establishing effective privacy and separation between customers is also key. Many cloud data platforms struggle at scale resulting in ugly workarounds. Yellowbrick customers have the confidence to deliver B2B applications at scale with confidence that the platform will continue to serve them cost effectively, even as they accelerate growth.

Scale-out SQL databases have hard limitations on the number of concurrent queries and the number of databases or schema objects that can be supported. Supporting large numbers of concurrent queries against increasingly large data sets can become prohibitively expensive and require architecturally ugly workarounds, such as running multiple instances or copies of data.

The largest instances of Yellowbrick have hundreds of compute nodes and thousands of logged in users executing hundreds of concurrent queries, ranging from short-running queries that complete in milliseconds through to multi-hour reports. This article explains how Yellowbrick manages concurrency and allows cost-effective scaling.

Innovative Architecture



Each Yellowbrick instance persists columnar data user data directly onto object storage for durability and availability. Each instance can house up to 999 user-created databases. The database is the topmost hierarchical object for organizing and storing SQL objects such as schemas, tables, views, and stored procedures. Up to 100,000 tables are supported per instance.

To execute queries, stateless elastic compute clusters are provisioned. Each contains a number of nodes, which do compute on the stored data. Each node has a high-performance cache, storing a local copy of the data persisted on object storage. Compute clusters can be dynamically created, resized, suspended / resumed, and destroyed. The more nodes and vCPUs present in a compute cluster, the faster it executes queries, but the more it costs to operate.

Each compute cluster can handle up to 150 concurrent queries. Up to 3,000 compute clusters can be provisioned per instance, with up to 3,000 nodes in total across them.

Queries submitted to a compute cluster pass through the workload manager and cluster load balancer. The workload manager is responsible for prioritizing queries, governing how many resources a query can use, and making sure that long-running or poorly written queries don't get in the way of short-running, tactical ones. The cluster load balancer provides a way of assigning queries to a running compute cluster with the least load.

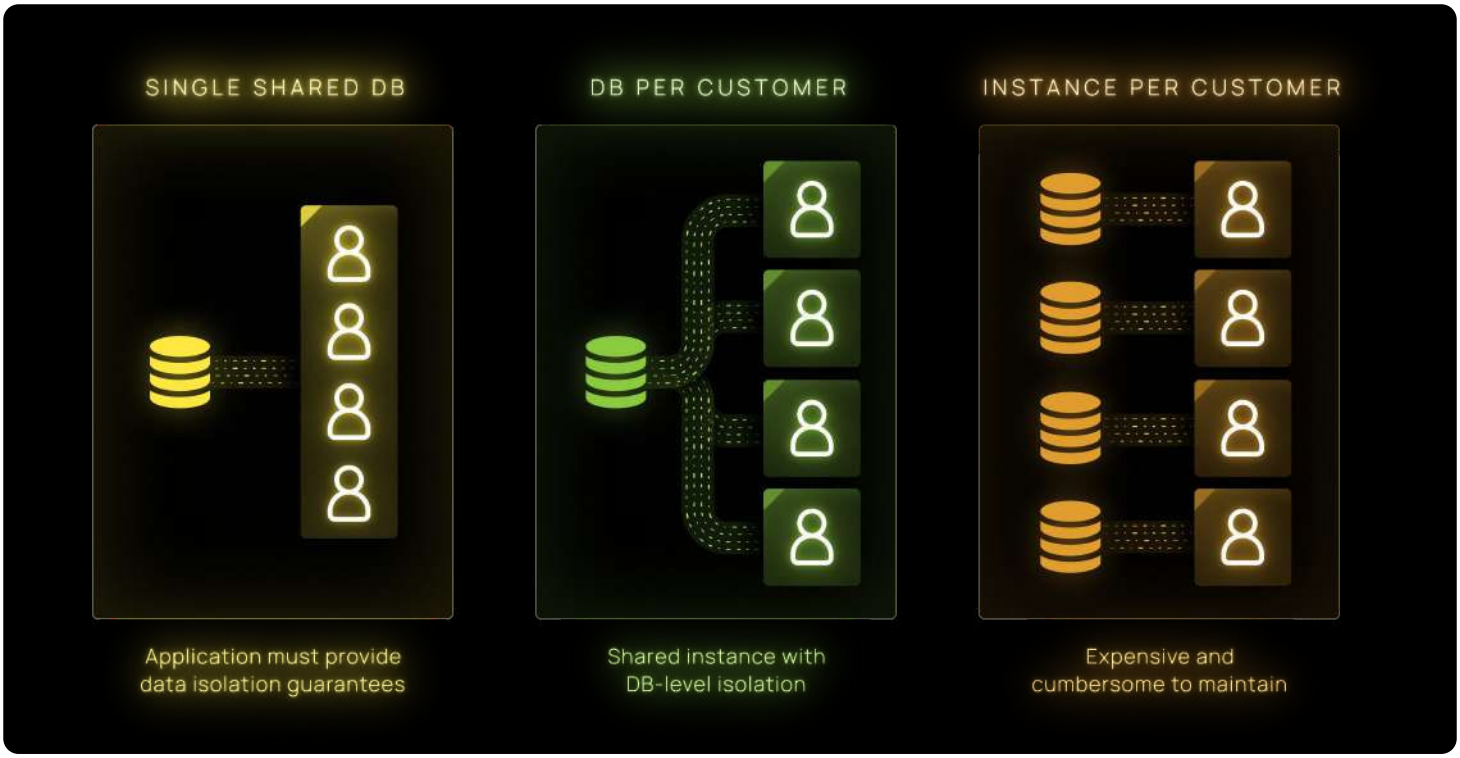
Up to 10,000 users can simultaneously connect to a Yellowbrick instance. There is effectively no limit on the maximum number of concurrent queries or the achievable Queries per Second (QPS). In practice, due to resource constraints, hundreds of concurrent queries executing at hundreds of QPS is a reasonable ceiling for large installations.

Implementing Multi-Tenant Applications

Databases are isolated containers to which permission must be explicitly granted to access the objects and data stored inside. When connecting to Yellowbrick, one explicitly connects to a named database to execute queries. Yellowbrick also supports cross-database queries if necessary.

There are a couple of approaches by which application developers can approach multi-tenancy in their applications: In a more complex and error-prone but flexible approach, all customers' data is stored in one database by intermingling records from different customers in the same tables and enforcing visibility on a row-by-row basis using a discriminator column ("tenant ID") enforced by views or other mechanisms. Such architectures have to be implemented extremely careful to avoid leakage of other customers' data; such leaks still happen often in production SaaS applications.

A simpler approach is to just provision one isolated database per customer.



This lessens the burden on the application developer and provides a simpler, more secure approach. This is enabled by Yellowbrick's support for large number of isolated databases, and the workload manager for making sure that users' queries don't interfere with each other. Either approach will work well with Yellowbrick.

Executing Efficiently

Scaling in an efficient and cost-effective way requires high performance query execution. Yellowbrick has the most efficient database execution engine in the industry, which we call our Direct Data Accelerator® [link to paper here]: Custom implementations of storage and network device access, optimized for the different public cloud and on-premises hardware infrastructure, use kernel bypass techniques to move data to and from object storage, local NVMe cache and across nodes on the network as fast as possible with minimal CPU utilization. This allows all CPU to be used for processing data, not moving it around.

The execution engine requires no buffer cache and uses a patented approach to stream data directly from storage at in-memory rates, making query performance consistent even under heavy load. Actual SQL operators are implemented as fully compiled and optimized vectorized code using LLVM.

To effectively schedule hundreds of concurrent queries, the execution engine includes a custom cluster-aware task scheduler developed and patented by Yellowbrick. The task scheduler guarantees predictable, better-than-linear scaling on both small and large CPUs. Unlike other databases which only support small numbers of concurrent queries scheduled by Linux, Yellowbrick can efficiently schedule large numbers of concurrent queries even on small CPUs without them interfering with one another.

Scaling and partitioning workloads across clusters

Supporting multiple compute clusters allows both workload isolation and workload scaling. For example, certain ad-hoc query users or ETL functions can be assigned dedicated compute clusters to make sure their workloads do not take resources from other users, or end-of-month and end-of-quarter additional processing clusters can be suspended for weeks at a time to save money.

The cluster load balancer allows the fundamental building block of a compute cluster to be used for building dynamic, scalable applications. By assigning users to a set of clusters, the combination of the workload manager and load balancer will dynamically assign queries to the cluster which can get the job done fastest. This isn't a case of simple round-robin or least-load load balancing: The cluster load balancer is able to look at available query slots and resources within each cluster to make the optimal allocation on a per-query basis. Suspending and resuming clusters, and adding new ones, allows dynamic scaling of database compute capacity with no downtime or interruptions, completely driven through SQL.